12

NOTE DI SOFTWARE

Honeywell
Honeywell Information Systems Italia
SW 12 S011003 01
CIMINO MICHELE
HISI-DIREZIONE GENERALE MARKETING
VIA PIRELLI 32
20124MILANO
Ufficio Documentazione Gestione Mailing List
Via Tazzoli 6-**20154 MILANO**
Trattasi di trasporto di cancelleria-stampati-moduli-documenti
interni-mobili e macchine per ufficio ESONERATI DALL'OBBLIGO
DELLE BOLLE DI ACCOMPAGNAMENTO D.P.R. 6/10/78 n. 627
art. 4 n. 8 e Circolare Ministeriale n. 72 del 23/12/78 paragrafo 10

UNIVERSITA'
DEGLI STUDI DI MILANO
ISTITUTO DI CIBERNETICA

Honeywell
Honeywell Information Systems Italia

Note di software.
è un bollettino trimestrale, edito a cura del Centro di Ricerca e Progettazione della Honeywell Information Systems Italia e dell'Istituto di Cibernetica dell'Università di Milano.

Note di software.
intende costituire uno strumento per la rapida e informale circolazione di informazioni, idee ed esperienze nell'area della tecnologia del software. In particolare, intende stimolare il dibattito sulle metodologie orientate alla diminuzione del rapporto costo/qualità dei programmi, e sugli strumenti atti ad automatizzare l'attività di produzione del software in ogni suo aspetto.

La collaborazione a **Note di software** è libera.
In particolare, si sollecitano contributi nella forma di brevi monografie o bibliografie essenziali commentate sui seguenti temi: tecniche e strumenti per la programmazione strutturata e modulare, metodologie e strumenti per il testing e il debugging dei programmi, documentazione del software, aspetti organizzativi e gestionali nella costruzione di sistemi software di grandi dimensioni.

Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Comitato di Redazione (HISI - Via ai Laboratori Olivetti - Pregnana Milanese), o di inviare direttamente il dattiloscritto in triplice copia.
I contributi non dovrebbero superare le quattromila parole.

I lavori accettati per la pubblicazione non vengono revisionati da un comitato tecnico. Pertanto, ogni contributo pubblicato riflette le opinioni personali dei suoi autori, e non necessariamente quelle del Comitato di Redazione.
Eventuali revisioni saranno effettuate di comune accordo fra il Comitato di Redazione e i proponenti.

Note di software.
viene distribuito a chi ne faccia richiesta alla redazione.

Direttore responsabile: Paolo Ghelfi
Comitato di Redazione: Roberto Polillo - Wanda Anselmetti
Redazione: Franco Soresini

12

FPDM-115

Ottobre · Dicembre 1979

NOTE DI SOFTWARE

Indice:

QUESTO NUMERO .... pag. 1

CONTRIBUTI TECNICI:

— ALCUNI LINGUAGGI A CONFRONTO, di G. Castelli ” 2

— LA GENERAZIONE DI RAPPORTI DA BASI DI DATI RELAZIONALI, di A. Albano, A. Marietti, M. E. Occhiuto, R. Orsini ” 14

— UN SISTEMA AD INDICE A DUE LIVELLI, di Le Van Huu ” 24

— SUR LA PROGRAMMATION RATIONNELLE DES ALGORITHMES NUMERIQUES, di A. Bossavit, B. Meyer ” 41

AVVENIMENTI:

— NOTE A MARGINE DEL WORKSHOP “STANDARDS FOR SYSTEM ANALYSIS”, Roma, marzo 1980 ” 50

IL SEGNALIBRO ” 55

Autorizzazione del Tribunale di Milano n. 87 del 23 febbraio 1977

QUESTO NUMERO ...

I linguaggi di programmazione, la generazione automatica di 'reports', i sistemi di generazione di indici e la programmazione strutturata di programmi di tipo numerico sono i temi trattati in questo dodicesimo numero di NOTE DI SOFTWARE.

Nel primo articolo, G. Castelli pone a confronto alcuni dei più recenti linguaggi di programmazione, derivati dal Pascal. I linguaggi trattati sono, oltre al Pascal stesso: Modula, Concurrent Pascal, Modula 2, Lis, Tartan ed Ada. Per ogni linguaggio viene data una sintetica 'scheda', destinata a fornire al lettore un primo orientamento sulle principali caratteristiche del linguaggio, e sui rapporti con il linguaggio progenitore. Per ogni linguaggio viene, inoltre, allegato un esempio di programma.

La possibilità di generare 'reports' organizzati in una forma che ne faciliti la comprensione e, nello stesso tempo, richieda all'utente un lavoro di specifica ridotto al minimo è il tema trattato nel secondo lavoro.
In esso, A. Albano, A. Marietti, M.E. Occhiuto e R. Orsini presentano un semplice linguaggio per la specificazione di 'reports' estratti da differenti insiemi di dati. L'approccio proposto, anche se utilizzabile in ambienti differenti, viene presentato per un sistema di basi di dati di tipo relazionale.

Le Van Huu presenta, nel terzo articolo, il KWOC, un sistema automatico di creazione di indici, utile per la catalogazione di vari tipi di documenti. Dopo una breve descrizione del sistema (opportunamente esteso), vengono mostrati alcuni esempi del suo uso in applicazioni differenti, e vengono esposte alcune considerazioni relative alla distribuzione statistica delle parole chiave per l'individuazione dei documenti.

Presentiamo, infine, un lavoro di taglio didattico, in cui A. Bossavit e B. Meyer illustrano, sviluppando un esempio, l'importanza dei metodi di dimostrazione di correttezza e dello sviluppo top-down nella costruzione di programmi di tipo numerico.

Questo numero contiene, infine, l'indice di tutti i lavori pubblicati, nei suoi tre anni di vita, da NOTE DI SOFTWARE.

La Redazione

CONTRIBUTI TECNICI

ALCUNI LINGUAGGI A CONFRONTO

G. Castelli
Istituto di Cibernetica - Università di Milano

Riassunto

È tracciata un'analisi comparativa dei linguaggi più strettamente derivati da Pascal e del loro progenitore, con particolare riguardo alla programmazione di sistema.
I linguaggi coinvolti sono: Pascal, Modula, Concurrent Pascal, Modula 2, Lis, Tartan e Ada.

Abstract

A comparison is made of some languages derived from Pascal, and of Pascal itself, with particular regard to system programming.
The examined languages are: Pascal, Modula, Concurrent Pascal, Modula 2, Lis, Tartan and Ada.

1. INTRODUZIONE

A fronte di metodologie rivolte ad indirizzare concettualmente la progettazione e la realizzazione di sistemi software occorre fornire strumenti implementativi adeguati. I linguaggi di programmazione, sebbene coinvolgano solo una fase abbastanza breve dell'intero ciclo produttivo, rivestono un'importanza concettuale e pratica rilevante. Legati agli aspetti pratici della questione sono i problemi della riduzione dei costi del software. La soluzione di tali problemi si ha quando, assunto corretto il disegno di un programma o di una procedura, è possibile garantire un elevato grado di affidabilità, portabilità, manutenibilità (modificabilità) e correttezza sin dalle prime fasi della stesura del codice e dalle prime prove in macchina.

L'aspetto concettuale riguarda l'estensione dei linguaggi da strumenti per "istruire" una macchina ad oggetti che creino in maniera naturale un ambiente adatto alla soluzione di classi di problemi.
Al di là dunque delle caratteristiche che rendono un linguaggio più attraente di un altro, un aspetto rilevante della questione riguarda la facilità di apprendimento e di uso.
Nei linguaggi moderni si è giunti alla scelta di dare all'utente un insieme di primitive ridotto, siano queste strutture di dati o di controllo, sulle quali costruire strutture flessibili e comunque complicate, eliminando il caos e il gigantismo di alcuni anni fa.

La rassegna che è presentata coinvolge un insieme di linguaggi che in qualche modo rappresenta l'evoluzione dei concetti esposti negli ultimi dieci anni.
Ovviamente sono rimasti esclusi numerosi linguaggi: è inevitabile in una rassegna di dimensioni contenute e che voglia seguire un tema.

Un altro lavoro di rassegna è comunque presente in letteratura [1].

2. ORGANIZZAZIONE DEL LAVORO

La rassegna è costituita di tre fasi:

- un riassunto in forma tabellare delle caratteristiche generali di tutti i linguaggi coinvolti
- una descrizione verbale espansa di tali caratteristiche e delle finalità del linguaggio
- un esempio che evidenzia le caratteristiche del linguaggio.

Per quest'ultimo caso non è stato scelto un unico programma da tradurre nei diversi linguaggi, ma ne sono stati scelti ad hoc, per evidenziare le caratteristiche proprie di ciascun linguaggio, non essendo spesso possibile realizzare un significativo confronto diretto. Quando possibile gli esempi sono stati tratti dagli stessi lavori di presentazione dei linguaggi nel tentativo di cogliere aspetti eventualmente sfuggiti nella sintesi di queste note.

SPECIFICHE DI TARGA	PASCAL	MODULA	MODULA 2	CONC. PASCAL	LIS	TARTAN	ADA
ROOT LANGUAGE	ALGOL 60	PASCAL	MODULA	PASCAL	PASCAL	PASCAL	PASCAL
SINTASSI	BNF	BNF	BNF	BNF	BNF	BNF	BNF
SEMANTICA	ASSIOMATICA	INFORMALE	INFORMALE	INFORMALE	INFORMALE	INFORMALE	INFORMALE
ORIENTATO A	GEN. PURP.	SYST. PROG.	SYST. PROG.	SYST. PROG.	SYST. PROG.	GEN. PURP.	GEN. PURP.
LINGUAGGIO							
ALLOCAZIONE MEMORIA	STACK/HEAP	STACK	STACK/HEAP	STACK/HEAP	STACK/HEAP	STACK/HEAP	STACK/HEAP
TIPO INTEGER	SI	SI	SI	SI	SI	SI	SI
REAL	SI	NO	NO	SI	SI	SI	SI
CHAR	SI	SI	SI	SI	SI	SI	SI
STRING	NO	NO	NO	NO	NO	NO	NO
BOOLEAN	SI	SI	SI	SI	SI	SI	SI
SUBRANGE	SI	SI	SI	SI	SI	SI	SI
ENUMERATION	SI	SI	SI	SI	SI	SI	SI
DOUBLE INT.	NO	NO	NO	NO	NO	NO	NO
DOUBLE REAL	SI	NO	NO	NO	SI	SI	SI
POINTER	SI	NO	SI	SI	SI	SI	SI
ARRAY	SI	SI	SI	SI	SI	SI	SI
RECORD	SI	SI	SI	SI	SI	SI	SI
SET	SI	SI	SI	SI	SI	SI	NO
VARIANT RECORD	SI	NO	SI	NO	SI	SI	SI
USER TYPE DEF.	SI	SI	SI	SI	SI	SI	SI
DATA ABSTRACTION	NO	SI	SI	SI	SI	SI	SI
IF THEN ELSE	SI	SI	SI	SI	SI	SI	SI
SCELTE MULTIPLE	NO	SI	SI	NO	SI	SI	SI
CASE	SI	SI	SI	SI	SI	SI	SI
ZAHN	NO	NO	NO	NO	NO	NO	NO
DO WHILE	SI	SI	SI	SI	SI	SI	SI
REPEAT UNTIL	SI	SI	SI	SI	SI	SI	SI
LOOP EXIT	NO(1)	SI	SI	NO	SI	NO	SI
DO VARYING	SI	NO	SI	SI	SI	SI	SI
GOTO	SI	SI	NO	NO	NO	SI	SI
MONITOR/GESTIONE CONC.	NO	SI	SI	SI	SI	SI	SI
EXCEPTION HANDLING	NO	NO	NO	NO	NO	SI	SI
CALL BY REFERENCE	SI	SI	SI	SI	SI	SI	SI
" " VALUE	SI	SI	SI	SI	SI	SI	SI
" " NAME	NO	NO	NO	NO	NO	NO	NO
IMP/EXP SIMBOLI	NO	SI	SI	SI	SI	SI	SI
I/O FACILITIES							
FILES	SI	NO	NO	NO	NO	SI	SI
SEQUENTIAL	SI	NO	NO	NO	NO	?	SI
INDEXED	NO	NO	NO	NO	NO	?	NO
DIRECT	NO(1)	NO	NO	NO	NO	?	SI
FORMAT	SI	NO	NO	NO	NO	?	SI
VARIE							
MACHINE DEPENDENCIES	NO	SI	SI	NO	NO	NO	SI
BIT VISIBILITY	NO	SI	SI	NO	SI	NO	NO
APPRENDIBILITÀ(4)	F	M	M	M	M	M	M
STANDARD INTERNAZ.	NO(2)	NO	NO	NO	NO	NO	NO
INTEGRAZIONE CON SIST.							
COMPILAZIONI SEPARATE	NO(1)	NO	NO	NO	SI	SI	SI
LINK ALTRI LINGUAGGI	NO(1)	NO	NO	NO	SI	?	SI
IMPLEMENT. INTERPRET.	SI	NO	NO	SI	NO	?	NO
" COMPILATE	SI	SI	SI	NO	SI	?	SI
" FIRMWARE	SI	NO	NO	NO	NO	?	NO
SPERIMENTATO	SI	SI	NO	NO	NO	?	NO
DEBUG FACILITIES	NO(1)	NO	?	NO	?	?	?

NOTE: (1) PRESENTI IN ALCUNE IMPLEMENTAZIONI
(2) IN CORSO DI DEFINIZIONE
(3) IN UNA VARIANTE PARTICOLARE
(4) F = FACILE; M = MEDIA

3. PASCAL

Pascal è un linguaggio della vecchia generazione. È stato infatti disegnato sul finire degli anni sessanta e rappresenta un diretto successore dell'Algol 60. La pulizia di disegno, seppure con alcune ambiguità ben illustrate in letteratura, e la sua semplicità costituiscono le caratteristiche dominanti.

Pensato inizialmente come strumento didattico e per la descrizione di algoritmi, ha raggiunto oggi anche il ruolo di linguaggio per la programmazione di sistemi, pur se privo di costrutti concorrenti.

Concetti

Rivisto oggi non contiene particolari raffinatezze. Aderendo alle indicazioni ormai storiche di [2], contiene numerosi tipi di dati raggruppabili a piacere dall'utente sino ad ottenere strutture molto complesse. La tipizzazione delle variabili permette controlli estensivi a compile-time. Dunque ogni oggetto deve essere dichiarato prima di essere usato.

Verificabilità

Una definizione assiomatica della semantica [3], seppure incompleta, permette la realizzazione di prove di correttezza.

Strutture dei programmi

Un programma Pascal è strutturato a blocchi, e ciascun blocco è costituito da una procedura o da una funzione. Queste possono essere comunque nodificate, a meno di restrizioni implementative. È ammessa l'attivazione ricorsiva di procedure o funzioni. La memoria è quindi gestita a stack. I parametri sono passati per valore o per indirizzo, ed è ammesso anche il passaggio di procedure o funzioni come parametri. Esiste un'area di memoria (heap) destinata a raccogliere le variabili puntate allocate e disallocate mediante le procedure standard **new** e **release (dispose)**.

Strutture di controllo

Sono le classifiche **if then else, for, while do, repeat until, case, goto e with**. Quest'ultima è dedicata all'accesso dei campi dei record senza richiedere prefissi.

Dati astratti

Non esistono strumenti per realizzarli. Solo le procedure e le funzioni definiscono oggetti astratti in senso lato.

Altre note

La descrizione contempla le caratteristiche essenziali del linguaggio, secondo lo standard definito de facto dal riferimento sottoindicato. Un gran numero di implementazioni esistenti ha esteso sia la sintassi, sia l'operabilità del linguaggio, ammettendo compilazioni separate, linkaggi con programmi scritti in altri inguagi, ecc.

Riferimento

K. Jensen, N. Wirth - Pascal User Manual and Report - Springer Verlag

Esempio

È introdotta una lista ricorsiva allo scopo di leggere stringhe di caratteri e di stamparle nello stesso ordine (fig. 1)

```
program copylist (input, output):
    type
        link = ↑ object;
        object =
            record
                next : link;
                data : char
            end;
    var
        base : link;
procedure append (var ptr : link);
    begin
        if ptr = nil
            then
                begin
                    new (ptr);
                    ptr ↑ . next : = nil;
                    read (ptr ↑ . data)
                end
            else append (ptr ↑ . next)
    end; { append }
procedure
    begin
        if ptr = nil
            then
                begin
                    write (ptr ↑ . data);
                    writelist (ptr ↑ . next)
                end
    end; { writelist }
begin { copylist }
    base : = nil;
    while not eof do
        append (base);
    writelist (base);
end.    { copylist }
```

Fig. 1

4. MODULA

Modula è un linguaggio per la programmazione concorrente e real-time. È stato pensato di dimensioni volutamente molto contenute per essere facilmente implementato su mini-calcolatori. La sua sintassi è largamente derivata da quella di Pascal, seppure con alcune restrizioni da un lato ed estensioni dall'altro (facilities di multiprogrammazione). Sviluppato dal gruppo di Wirth, è stato presentato nel 1977.

Concetti

Elemento basico del linguaggio è il modulo. A questo è demandato il controllo di superare la struttura a blocchi realizzando la possibilità di disporre di uno strumento per costruire strutture di dati astratti e information hiding. È uno strumento più flessibile e generale di quelli disponibili in altri linguaggi.

I processi sono algoritmi sequenziali, eseguibili concorrentemente ad altri. Un processo viene attivato da una chiamata di processo, formalmente analoga ad una chiamata di procedura.

I moduli d'interfaccia garantiscono la mutua esclusione sugli oggetti in essi contenuti da parte di processi concorrenti. Su tali oggetti è lecito operare solo tramite procedure anch'esse locali ai moduli d'interfaccia. I processi comunicano tra loro via procedure standard **wait, send, awaited** che operano su variabili del tipo predefinito **signal**. Interfacce per drivers sono definite da device modules la cui struttura è tuttavia molto influenzata dalle caratteristiche della macchina (PDP 11 nella prima implementazione).

Verificabilità

La semantica informale rende non applicabili intenti di verificabilità.

Struttura dei programmi

Un programma Modula è costituito da una collezione di procedure o funzioni e di dati. Le procedure possono essere attivate ricorsivamente e hanno due tipi di parametri: variabili e costanti.
Le procedure costituiscono dei blocchi.

Allo scopo di estendere la vita di oggetti locali ad una procedura dopo la sua terminazione o di rendere condivisibili tra procedure diverse oggetti comuni e nascosti, è stato introdotto il modulo. Il modulo può essere locale ad una procedura ed in tal caso gli oggetti dichiarati al suo interno (variabili, tipi, costanti, procedure) cominciano ad esistere quando viene attivata la procedura a cui il modulo è locale. La visibilità degli oggetti all'interno ed all'esterno è regolata da due liste: la define list e la use list.
La prima stabilisce quali oggetti dichiarati all'interno possono essere usati all'esterno, la seconda il viceversa.

Strutture di controllo

Esistono strutture di selezione (**case, if then elsif**) e di iterazione (**repeat until, while do, loop exit**).

Dati astratti

La struttura costituita dal modulo permette una agevole costruzione di enti astratti, realizzando anche funzioni di information hiding.

Altre note

Oltre ai tipi predefiniti "classici": **integer, boolean, char** (qui manca il tipo **real**), è stato introdotto un nuovo tipo detto **bits**, definito come un **array (.O..n.) of boolean** con uguale al numero di bit diminuito di uno di una parola.

Sono assenti tipi complessi quali i files e le relative operazioni di I/O a meno di una procedura primitiva **DOIO**.
Ciò perchè Modula si presenta come uno strumento per l'implementazione di tali oggetti più che un loro utilizzatore.

Riferimento

N. Wirth - Modula: a language for modular multiprogramming Software and Experience - n. 1, vol. 7, 1977

Esempio

Viene descritta una tabella di gestione di tracce di un disco, e le protegge da accessi illegali (fig. 2).

```
module trackreservation;

  define newtrack, returntrack;
  const m = 64;w = 16; (*there are m*w tracks*)
  var i : integer;
    reserved : array 0 : 63 of bits;

  procedure newtrack : integer
   (*reserves a new track, yelds its index as function
     result, if a free track is found, and - 1 otherwise*)
   var i, j : integer; found : Boolean;
  begin found : = false; i : = m;
     repeat dec (i); j : = w;
          repeat dec (j);
              if not reserved [ i, j ] then found : = true end
          until found or (j = O)
     until found or (i = O);
     if found then newtrack: = i*w + j;reserved [i, j]: = true
              else newtrack: = - 1 end
  end newtrack;

  procedure returntrack (k : integer);
  begin (*assume O < = k < m*w*)
     reserved [ k div w, k mod w ] : = false
  end returntrack;

begin i : = m; (*mark all tracks free*)
     repeat dec (i); reserved [ i ] : = [ ]
     until i = O
end trackreservation
```

Fig. 2

5. CONCURRENT PASCAL

Concurrent Pascal è stato disegnato nel 1975 da P. Brinch Hansen e rappresenta il tentativo di definire uno strumento adatto a implementare i concetti di regione critica e di monitor. L'area di applicazione è dunque tipicamente quella rivolta alla soluzione di problemi di concorrenza.

In particolare, l'applicazione più rilevante del linguaggio è stato Solo, un sistema operativo monoutente, interamente realizzato in Concurrent Pascal.

La sintassi è largamente derivata da Pascal, con l'aggiunta di tipi di dati astratti.

Concetti

Gli elementi fondamentali del linguaggio sono costituiti da tre tipi di dati astratti: classi, processi e monitors. Questi si presentano sintatticamente come dei tipi predefiniti che vengono riempiti di dati e codice ad essi locali.
Come per qualunque tipo di dato è dunque possibile instanziare un numero qualsiasi di variabili.

Una classe è una componente di sistema che non può essere chiamata simultaneamente da altre. Il suo scopo è dunque quello di nascondere gli oggetti in essa contenuti.

Un monitor è una componente di sistema che definisce un insieme di oggetti condivisibili. Realizza il concetto di regione critica, garantendo la mutua esclusione sugli oggetti contenuti al suo interno.

Un processo definisce un programma sequenziale che può interagire con altri processi solo all'interno dei monitors.

Verificabilità

Non si pone come uno degli scopi del linguaggio. Sforzi sono comunque stati fatti per la correttezza dei monitors [4].

Struttura dei programmi

La struttura a blocchi è mantenuta all'interno di ciascun tipo di dato astratto. I filtri alla visibilità sono costituiti dalle liste di parametri che appaiono nelle dichiarazioni delle componenti di sistema. Questi parametri, rappresentano anche i diritti che una componente di sistema può vantare nell'accederne altre.

Parametri di procedure e funzioni possono essere passati per indirizzo e per valore.

Strutture di controllo

Sono quelle di Pascal, estese con un ciclo senza fine (**cycle**).

Il controllo della sincronizzazione tra processi è realizzato sotto il controllo del programmatore, mediante le procedure **delay** e **continue** che operano esclusivamente su variabili del tipo predefinito **queue**.

Dati astratti

La capacità di astrazione sono elevate grazie ai tipi predefiniti monitor, classe e processo. La loro concatenazione è possibile e la legalità degli accessi è verificata a compile-time.

Altre note

Esistono due livelli di sincronizzazione: una stabilità dai monitor che garantiscono la mutua esclusione ma soggetta ad una commutazione del controllo tra i processi dipendenti dall'implementazione (17 msec. nell'implementazione di Brinch Hansen), e una operante su code definite dall'utente e operabili via **delay**, **continue**, ed **empty**.

È possibile anche un'eliminazione esplicita dei controlli sui tipi delle variabili.

Per evitare condizioni di stallo all'interno dei monitors è stata eliminata la ricorsione.

Riferimento

P. Brinch Hansen - Concurrent Pascal Report - Institute of Technology - University of California.

Esempio

Due processi Input e Output comunicano tra loro attraverso un buffer sul quale è garantita la mutua esclusione.
Si assume l'esistenza di una procedura **read** e una **write** (introdotte per semplicità ma non esistenti in Concurrent Pascal) (fig. 3).

```
type block = . . . .

type iobuffer = monitor;
  var buffer: block; inuse : boolean;
              free, loaded : queue;

  procedure entry deliver (in : block);
    begin
     if inuse then delay (free);
        buffer : = in;
        inuse : = true;
        continue (loaded)
    end;

  procedure entry retrieve (out : block);
  begin
      if not inuse then delay (loaded);
      out : = buffer;
      inuse : = false;
      continue (free)
    end;

  begin inuse : = false end;

type inputproc = process (communication : iobuffer);
  var in : block;
  begin
    cycle read (in) ; communication . deliver (in) end
  end;

type outputproc = process (communication : iobuffer);
  var out : block;
  begin
    cycle write (out); communication . retrieve (out) end
  end;

var input : inputproc;
    output : outputproc;
    comm : iobuffer;
begin
    init comm, input (comm), output (comm) end.
```

Fig. 3

6. MODULA 2

Modula 2 è un linguaggio general purpose particolarmente rivolto all'implementazione di sistemi, in particolare su macchine di piccole dimensioni. Per tale motivo le dimensioni del linguaggio sono state mantenute contenute. Modula 2 rappresenta l'evoluzione di Modula, seguendolo sulla strada cara e Wirth della compattezza e della pulizia di disegno. La rassegna qui esposta rende conto dello stato dell'arte al dicembre 1978.

Concetti

Sono derivati in gran parte da Modula e Pascal, con la riscoperta di altri più antichi, quali le coroutines, anche se rivestiti di un abito sintattico più moderno.

I concetti fondamentali riguardano:

- i moduli: porzioni di dati e codice con regole di visibilità esplicite (import, export), trascendenti quelle delle strutture
- processi: da interpretare come coroutines

Verificabilità

Non è stato fatto nessuno sforzo per orientare il linguaggio in tale direzione.

Struttura dei programmi

I programmi sono insiemi di dati e codice la cui visibilità dipende dalla struttura imposta dai blocchi. Ai concetti fondamentali di procedure e funzioni si è sovrapposto il modulo al fine di alterare esplicitamente la visibilità.

Il passaggio di parametri avviene per indirizzo e per valore (default). La ricorsione è possibile.

Strutture di controllo

Largamente derivate dalla versione precedente, dispone di strutture di selezione **if then else (elsif), case**, di iterazione **while do, repeat until, for, loop exit**. Sono state abolite le primitive di sincronizzazione **wait, send, awaited** e sono state introdotte le procedure **newprocess** per l'allocazione di istanze di processi e **transfer** per l'attivazione di un processo con la relativa sospensione di quello in corso. Ai processi attivati è assegnata un'area di lavoro di dimensione stabilita dall'utente e l'indirizzo d'inizio di questa.

Dati astratti

Vale in generale quanto esposto per Modula, ed anzi la situazione è migliorata grazie alla presenza di moduli "trasparenti" ed "opachi".

Esempio

Un processo rappresentato dal programma principale colloquia con un altro, generato da Call Partner (fig. 4).

```
module Conversation;

  from system import
     process, proc, newprocess, transfer;
  export CreatePartner, Reply;
  var spr : process; (*suspended process*)
      msg: cardinal;
      wsp: array 0 . . 99 of word; (*workspace*)

  procedure Call Partner (P: proc) : cardinal;
  begin newprocess (P, adr (wsp), size (wsp), spr);
     transfer (spr, spr); return msg
  end Call Partner;

  procedure Reply (x : cardinal) : cardinal;
  begin msg : = x; transfer (spr, spr); return msg
  end Reply;

end Conversation
```

Fig. 4

7. LIS

Lis è un linguaggio rivolto all'implementazione di sistemi realizzato intorno al 1976 dalla CII. Può considerarsi il più diretto predecessore di Ada, sia per il gruppo che l'ha realizzato sia per la forma assunta dai concetti implementati.

Concetti

Lis è legato storicamente alla corrente di pensiero che ha generato Simula e Pascal. Gli aspetti innovativi riguardano il grande peso dato agli aspetti di manutenibilità dei programmi, realizzati mediante la distinzione tra la fase di definizione di dati e algoritmi e una fase di implementazione dei dati stessi. In tal modo, gli algoritmi possono essere formulati in relazione alle caratteristiche semantiche dei dati e sono invarianti rispetto alle alterazioni implementative di questi.

Gli schemi di implementazione servono anche per specificare aspetti dipendenti dalla macchina e quando si usino sottoprogrammi in altri linguaggi. Tuttavia errori in questa fase sono estremamente pericolosi non essendo rilevabili nè a compile-time nè dal supporto run-time.

È possibile inoltre fare ricorso a compilazioni separate. Gli oggetti che le permettono sono detti segmenti e partizioni. Esistono segmenti di dati e codice e partizioni dello stesso tipo. I segmenti sono organizzati in una maniera gerarchica che definisce sia delle regole di visibilità, sia quali oggetti devono essere ricompilati in caso di alterazioni. Le partizioni definiscono delle suddivisioni dei segmenti e introducono raffinamenti alle regole di visibilità.

Verificabilità

Non si pone come uno dei fini del linguaggio.

Struttura dei programmi

Un programma Lis è una collezione di unità di compilazione. Un'unità di compilazione può consistere di:

- un sottoprogramma (segmento di programma)
- un insieme di dichiarazioni di dati di un sottoprogramma (segmento di dati)
- un sottoinsieme di un insieme di dichiarazioni (partizioni di dati)
- un sottoinsieme di corpi di un sottoprogramma (partizione di programma)

La struttura generale è a blocchi. I sottoprogrammi sono: procedure e funzioni (sottoprogrammi ricorsivi), azioni (sottoprogrammi non ricorsivi) e coroutines. I parametri sono passati per valore ed indirizzo. Non sono ammessi side effect all'interno delle funzioni. Esistono inoltre opportune variabili, dette connettori, che identificano sottoprogrammi, la cui invocazione determina l'esecuzione del sottoprogramma associato.

Strutture di controllo

Sono presenti: **if then else (elsif), case, repeat until, while do, loop exit.**

Dati astratti

Benchè non esista la possibilità di creare strutture astratte in modo esplicito, due caratteristiche del linguaggio costituiscono buoni surrogati da un punto di vista funzionale: la distinzione tra definizione e implementazione dei dati e la presenza dei connettori.

Altre note

È esplicitamente indicata nelle caratteristiche del linguaggio la possibilità di ottenere il pretty printing implicito dei programmi.

Riferimento

CII - The system implementation language Lis -Reference manual.

Esempio

È illustrata la suddivisione di un programma in partizioni o segmenti e le conseguenti regole di visibilità degli oggetti (fig. 5).

```
use data main;
data segment semantics;
   genecode : partition;
   treatparameters : partition;
end

use data main semantics;
data partition treatparameters;
   simpleparameter, formalparameter action
end;

use data main semantics;
data partition gencode;
   genadd, genmulti, gendiv, gensub: action
   type instruction
      plex
      . . . . .
      end;
end;

use data main, semantics, gencode;
program partition gencode;
   bufcode array (1..100) of instruction;
   program genadd;
   begin
   . . . . .
   end;
   program gensub;
   begin
   . . . . .
   end;
. . . . .
end;

use data main, semantics, gencode, treatparameters;
program segment semantics;
begin
   . . . . .
   simpleparameter;
   . . . .
   genadd;
   . . . .
end;
```

Fig. 5

8. TARTAN

Tartan è un linguaggio sperimentale, progettato in accordo con l'Ironman requirement del DoD. È stato sviluppato per dimostrare come fosse possibile disegnare un linguaggio estremamente semplice che soddisfacesse quei requirements. Presumibilmente non sarà mai operativo, ma viene riportato a causa del contesto in cui è nato e per confronto con il vincitore del concorso, Ada.

Concetti

Anche Tartan è un derivato di Pascal. Ne induce quindi la struttura a blocchi modificata da moduli.

Entrambi definiscono le regole statiche di visibilità legate alla struttura lessicale. Sono dunque definite regole implicite ed esplicite per la visibilità degli identificatori. In particolare, i blocchi importano automaticamente identificatori ma non sono in grado di esportarli in alcun modo; i moduli possono importare/esportare identificatori in modo esplicito.

La vita degli oggetti è invece regolata solo dai blocchi, mentre i moduli non possono ereditare l'ambiente che li circonda.

È possibile definire oggetti "generici", che possono cioè riassumere le caratteristiche di molti diversi.
È in seguito possibile definire le opportune istanze per questi oggetti.

Programmi concorrenti possono essere gestiti come processi e la loro comunicazione avviene tramite variabili "latch".

I processi possono essere attivati e bloccati mediante l'uso delle istruzioni **create, activate**. Il processo su cui queste operano può trovarsi in quattro stati diversi **mint, suspended, active, dead**. Esiste la possibilità di inserire asserzioni verificabili a run-time.

È infine possibile la gestione delle eccezioni.

Verificabilità

Non trattata.

Struttura dei programmi

I programmi sono costituiti da fasi di dichiarazioni e da frasi eseguibili. Le definizioni associano nomi a oggetti lessicali, le dichiarazioni associano nomi a valori e a variabili, le frasi eseguibili definiscono la sequenza di esecuzione. Esistono procedure e funzioni (ricorsive) alle quali è possibile passare parametri secondo quattro modalità: **var** (per indirizzo), **const** (per valore) **manifest** (espressioni valutate a compile-time e passate per valore), **result** (per indirizzo, ma con check rilassati).

Strutture di controllo

Oltre alle classiche **if then else (if), case, while do, for, goto** esistono frasi utilizzate per la gestione delle eccezioni: **signal, resignal, assert** e per la gestione dei processi: **create** e **active**.

Dati astratti

È presente il concetto di modulo, con caratteristiche pressochè identiche a quello di Modula.

Altre note

Per eliminare le ambiguità derivanti dall'assenza dei terminatori di costrutto e la propagazione di errori sintattici, i terminatori sono stati introdotti nel classico modo dell'inversione (**if . . . fi, case . . . esac**, ecc.).

Riferimento

M. Shaw, P. Hilfinger, W.A. Wulf - Tartan: language design for the Ironman Requirements - SIGPLAN Notices, vol. 13, n. 9, sept. 1978

Esempio

Due tipi alternativi di code sono rappresentati.
La prima è del tipo degli elementi da accodare, la seconda è una costante manifesta che seleziona quale rappresentazione utilizzare. Le code sono buffer circolari (fig. 6).

9. ADA

Ada è il vincitore del contratto bandito nel 1975 dal Dipartimento della Difesa USA (DoD) per la definizione di un linguaggio unificato da usarsi in sostituzione di un numero eccessivo di linguaggi, all'interno delle diverse armi. Anch'esso è un discendente di Pascal. È stato sviluppato come un linguaggio general purpose, che possiede anche strutture per la programmazione modulare real-time.

Concetti

Non esiste un elemento innovativo caratterizzante il linguaggio. Sono altresì state raccolte e consolidate molte delle indicazioni contenute nei linguaggi già esposti. Si ritrovano dunque strutture di controllo e di dati classiche e i ben noti concetti di tipizzazione delle variabili.

Anche in Ada si ritrova il concetto di modulo, visto sia come strumento rivolto all'astrazione, sia come strumento implementativo per una gestione semplificata delle compilazioni separate, sia come contenitore per creare un ambiente adatto alla gestione dei task.
Ritroviamo pertanto due tipi diversi di moduli: package e task.
Ogni modulo è cosituito da due parti: una fase di dichiarazione ed un corpo.

```
generic module queuedef [ T : type; F. num [ Fix, Flex] ]
   begin

   exports queue [ T ]                          ! type attribute Size
        clear, enq, deq, empty, full,           ! routines
        QOvfl:                                  ! exception
   module lst is listdef [ t ];
   module cbf is circularbuffers [ t ];
   type queue [ t ] (size:int) =
      variant manifest fx : enum [ fix, flex ] : = f
       [ on fix → circbuf [ t ] (size) on flex → list [ t ] ]
   exception QOvlf;

   proc clear ( result q : queue [ t ] );
      begin
      case f on fix → clear ( q(fix) ) on flex → clear ( q(flex) ) esac
      end;

   proc enq (var q : queue [ t ], const val : t);
      begin
      case f
         on fix → append ( q(fix), val ) [ on bufovfl → signal QOvfl ]
         on flex → insert ( last(q(flex)), val )
      esac
      end;

   proc deq ( var q : queue [ t ], result val : t );
      begin
      case f
         on fix → remove ( q(fix), val )
         on flex → begin val : = q(flex) . data; delete ( q(flex) ) end
      esac
      end;

   func empty ( q : queue [ t ] )e : boolean
      begin case f
                on fix → e : = empty ( q(fix) )
                on flex → e : = ( q(flex) = nil )
            esac
      end;

   func full ( q : queue [ t ] )e : boolean;
      begin
      case f on fix → e : = full ( q(fix) . fixrep ) on flex → e : = false
      esac
   end;          ! module queuedef
```

Fig. 6

L'importazione/esportazione di oggetti è esplicita. I task module comunicano tra loro mediante le frasi **entry**, **accept**, **select**, e **delay**.

È offerta anche la possibilità di gestire eccezioni, gestite comunque come condizioni di terminazione locale.

Verificabilità

Non trattata.

Struttura dei programmi

I programmi sono organizzati a blocchi, a cui si sovrappone il modulo.

In particolare è possibile costruire, attraverso i moduli, unità di compilazione opportune. In tal senso ci si avvicina ai concetti di modularizzazione propri di Lis.

È possibile anche la definizione e la ridefinizione degli operatori aritmetici.

```
task spooler is
   entry start-transfer (source, destination: in file-name);
end;

task body spooler is
   inf   : in-file;
   out   : out-file;
   c     : character
   procedure open-in-out (source, destination: in file-name) is
   begin
      open (inf, source);
      begin
         open (outf, destination);
      exception
         when file-name-error => close (inf); raise;
      end;
   end;

begin
   loop
   begin
      accept start-transfer (source, destination: in file-name) do
         open-in-out (source, destination);
      end;
      loop
         get (inf, c);
         put (outf, c);
      end loop;
   exception
      when end-of-file =>
         put (outf, eof);
         close (inf);
         close (outf);
      when file-name-error => null; -- restart main loop
   end;
   end loop;
end spooler;
```

Fig. 7

I parametri sono passati per indirizzo o per valore ed il numero dei parametri attuali può differire dal numero di quelli formali (formali > attuali). In tal caso vengono assunti valori di default.

Strutture di controllo

Sono: **if then else, case, loop** (elemento fondamentale di ciclo componibile con **for** e **while**), **exit, goto, assert, exception.**

Dati astratti

La capacità di definire strutture di dati astratti è presente grazie ai moduli. Questi non differiscono sostanzialmente da quelli di Modula o di Lis.

Altre note

Gli aspetti riguardanti i task meritano ancora due parole. I task comunicano tra loro mediante il concetto di rendez-vous. Anche questo non è un concetto nuovo, essendo stato introdotto da Conway nel 1963, ma è stato recentemente rivisto e generalizzato da P. Brinch Hansen [5] e da C.A.R. Hoare [6]. È tuttavia la prima implementazione in un linguaggio. Un esempio in fig. 7.

Riferimento

Preliminary Ada Reference Manual - SIGPLAN Notices - vol. 14, n. 6, giugno 1979

Esempio

Un task USER realizza un trasferimento di file tramite un task SPOOLER in modo asincrono. La procedura OPEN-IN-OUT apre i due files. Se un file è invalido, si verifica una condizione di eccezione (fig. 7).

10. BIBLIOGRAFIA

[1] F. De Remer, P. Levy (Editors), SUMMARY OF THE CHARACTERISTICS OF SEVERAL 'MODERN' PROGRAMMING LANGUAGES, SIGPLAN Notices Vol. 14, n. 5, Maggio 1979

[2] Dahl, Dijkstra, Hoare, STRUCTURED PROGRAMMING, Academic Press, 1972

[3] C.A.R. Hoare, N. Wirth, AN AXIOMATIC DEFINITION OF THE PROGRAMMING LANGUAGE PASCAL, ACTA Informatica vol. II, n. 4, 1973

[4] J.H. Howard, PROVING MONITORS, Communication ACM, vol. 19, n. 5, Maggio 1976

[5] P. Brinch Hansen, DISTRIBUTED PROCESSES: A CONCURRENT PROGRAMMING CONCEPT, Communications ACM, vol. 21, n. 11, Nov. 1978

[6] C.A.R. Hoare, COMMUNICATING SEQUENTIAL PROCESSES, Communications ACM, vol. 21, n. 8, agosto 1978

Ringraziamenti

Il presente lavoro è stato svolto nell'ambito del Communication Programming Project tra l'Istituto di Cibernetica della Università di Milano e l'Honeywell Information Systems Italia.

LA GENERAZIONE DI RAPPORTI DA BASI DI DATI RELAZIONALI

A. Albano(*), A. Marietti(°), M.E. Occhiuto(*), R. Orsini(*)

Riassunto

L'esigenza di produrre agevolmente rapporti, da uno o più insiemi di dati, organizzati in una forma che ne faciliti la comprensione e nello stesso tempo richieda all'utente una quantità di specifiche ridotta al minimo, è sentita anche per i moderni sistemi di gestione di basi di dati. Viene presentato un formalismo per dichiarare rapporti con la possibilità di estrarre ed intercalare registrazioni appartenenti ad insiemi diversi di dati.

Abstract

It is current need even for the modern data base management systems to generate reports starting from one or more sets of data. Reports should be organized in a format easy to understand and at the same time the user should be required a minimum amount of input specifications. A formalism is presented providing the capability for extracting and intermixing records originating from different sets of data.

1. INTRODUZIONE

In molte situazioni e in particolare nelle applicazioni gestionali, è sentita l'esigenza di produrre agevolmente rapporti (prospetti o tabulati) da uno o più insiemi di dati. I rapporti sono, nella maggior parte dei casi, una stampa di dati organizzata in un'**intestazione**, seguita da **righe di dettagli**, eventualmente interrotta in punti opportuni da **righe di totali** dei valori di alcuni attributi e da alcune **righe finali**. Se il rapporto si estende su più pagine, la stampa può essere prevista con una **numerazione** e la ripetizione di un'intestazione su ogni nuova pagina.

La possibilità di generare rapporti organizzati in una forma che ne faciliti la comprensione e nello stesso tempo richieda all'utente una quantità di specifiche ridotta al minimo è stata sempre offerta dai linguaggi di gestione di archivi prima (ad esempio con i Report Program Generators) e dai sistemi di basi di dati poi [7].

(*) Istituto di Scienze dell'Informazione dell'Università di Pisa
(°) Honeywell I.S.I. - Pregnana Milanese

Lavoro eseguito nell'ambito della collaborazione fra Honeywell I.S.I. e Istituto di Scienze dell'Informazione dell'Università di Pisa.

Le caratteristiche degli strumenti disponibili a tale scopo sono andate cambiando con l'evolversi della concezione dell'organizzazione degli archivi, delle modalità d'uso dei dati e con l'affermarsi dei sistemi di gestione di basi di dati.

Nei sistemi tradizionali, orientati ad elaborazioni a lotti, per generare rapporti il linguaggio di programmazione (tipicamente il COBOL o l'RPG) è esteso con opportuni comandi o dichiarazioni, sia per descrivere il formato del rapporto (numero di righe di stampa per pagina, la intestazione desiderata, il formato delle righe di dettaglio, la modalità di generazione di righe intermedie, ad esempio di totali parziali, ecc.), sia per attivare il modulo che si basa su queste descrizioni per generare il rapporto sui dati selezionati dall'utente. Quest'ultimo tipo di comandi, in generale, appaiono in un ciclo nel quale vengono preparati i dati delle righe di dettaglio scandendo opportuni archivi.

In queste applicazioni, inoltre, è frequente il caso di generazioni periodiche di rapporti per uso contabile o statistico che riguardano tutti i dati degli archivi interessati. Si pensi ad esempio alla generazione periodica dell'estratto conto dei clienti di una banca [6].

Con la diffusione dei sistemi interattivi l'importanza della generazione dei rapporti non diminuisce, ma viene richiesto di soddisfare anche altre esigenze:

— I rapporti appaiono su terminali video e spesso non interessano formati elaborati, imposti invece dai moduli prestampati quali il bollettino degli stipendi, fatture ecc. È sufficiente un formato standard che, ad esempio, oltre ai dati evidenzi i nomi degli attributi delle registrazioni dalle quali provengono, dia alcuni titoli e eviti all'utente di dare specifiche non essenziali quali i titoli, le spaziature, allineamenti, ecc.

— I dati sono in generale in quantità ridotta, perchè sono richiesti per esigenze più specifiche e che cambiano nel tempo, e vanno presentati con un formato variabile e non prevedibile in anticipo (ad esempio, può essere l'elenco dei clienti di una particolare filiale, degli ordini ricevuti ad una certa data, dei prodotti con determinato valore di costo o giacenza, ecc.).

— I dati appartengono anche ad archivi diversi e si deve avere la massima flessibilità nella scelta dell'ordine in cui vanno inseriti nel rapporto [3].

In questo lavoro considereremo questa seconda categoria di applicazioni. In particolare, viene proposto un metodo dichiarativo per richiedere la generazione di rapporti, con la possibilità di estrarre ed intercalare registrazioni appartenenti a insiemi diversi di dati.

Nei prossimi paragrafi, dopo la definizione degli obiettivi della proposta, viene presentato un metodo di dichiarazione dei rapporti e se ne mostra l'efficacia attraverso alcuni esempi.

È interessante notare che in sviluppi di recente introduzione, come il Sistema/38, il problema della generazione di rapporti, a partire da insiemi diversi di dati, è al centro dell'impostazione del sistema di gestione di basi di dati. Basti citare, a questo proposito, l'introduzione dei seguenti concetti e prodotti:

— archivi logici contenenti registrazioni provenienti da insiemi diversi di dati (archivi logici)

— programma di utilità "Query" per la stampa di rapporti a partire da archivi logici o fisici.

2. IMPOSTAZIONE DEL PROBLEMA

Il metodo che si propone per descrivere rapporti verrà presentato per un sistema di gestione di basi di dati relazionale, dato che per questi sistemi ne è molto sentita l'esigenza [5], ma, essendo sostanzialmente basato sui concetti di entità, proprietà e associazione, esso è facilmente utilizzabile anche per gli altri tipi di sistema.

Richiamiamo brevemente alcuni concetti fondamentali che stanno alla base del modello relazionale.

Una **relazione** R su n domini S_1, S_2, . . . , S_n, non necessariamente distinti, è un sottoinsieme del prodotto cartesiano $S_1X\ S_2X\ \ldots\ XS_n$, in altre parole consiste di un certo numero di tuple, ciascuna contenente n campi. Le tuple devono essere tutte distinte, quindi esiste un sottoinsieme di campi K di R detto **chiave** con la proprietà che in ciascuna tupla di R il valore di K identifica univocamente quella tupla; nessun campo, inoltre, può essere eliminato senza che la chiave perda la proprietà precedente.

Una **base di dati relazionale** è un insieme finito di relazioni variabili nel tempo. Le corrispondenze tra le entità sono date dai valori di opportuni attributi, si fa uso cioè di **chiavi esterne** che sono insiemi di attributi che assumono valori nel dominio della chiave primaria di un'altra relazione e che non costituiscono chiave primaria nella relazione in cui appaiono.

I linguaggi proposti per l'uso di basi di dati relazionali (come il SEQUEL o il Query by example) sono molto efficaci nell'esprimere condizioni complesse per l'estrazione dei dati, ma mal si prestano alla definizione di rapporti di una certa complessità.

Si riesce, infatti, facilmente ad ottenere la stampa dei valori di un insieme di attributi di una o più relazioni, ma il rapporto ottenibile è ancora l'immagine di una relazione, cioè ogni riga di stampa ha il medesimo formato. Questa forma di rapporto è adatta solo a casi molto semplici, mentre è molto sentita la necessità di descrivere rapporti in cui sia possibile intercalare con più modalità tuple di relazioni diverse, come ad esempio per far sì che una tupla "cliente" preceda le relative tuple "ordini" e ciascuna di queste le proprie "linee ordini" e così via.

La descrizione di un rapporto in questo contesto consiste di un insieme di frasi in un opportuno linguaggio, per specificare:

— Le relazioni che interessano;
— di ogni relazione, quali domini dovranno apparire nelle righe del rapporto e sotto quale intestazione;
— l'ordine nel quale devono essere presentati i dati;
— la modalità in cui vanno intercalati i dati provenienti da relazioni diverse fra le quali esiste una corrispondenza.

Vediamo, nell'ordine, la descrizione di questi requisiti, di complessità crescente, facendo riferimento alla base di dati di Fig. 1. Gli esempi che daremo sono scritti nella sintassi che verrà data nel seguito.

CLIENTI

Codice	Regione	Nome	Indirizzo
C100	Lazio	Albini	V.Nomentana Roma
C101	Toscana	Rossi	P. Guerrazzi Pisa
C102	Lombardia	Zanardi	V. Risorgimento Mi.
C110	Toscana	De Mita	V. Redi Firenze
C111	Toscana	Rossi	V. Polese Livorno
C120	Lombardia	Bossi	V. Tealdi Milano

ORDINI

Codice	Codice-cliente	Codice prodotto	Quantità	Data
O055	C101	P37	1000	8-3-77
O059	C111	P42	10	19-2-78
O091	C120	P50	45	21-4-78
O103	C110	P42	15	8-3-78
O120	C102	P37	500	11-1-78
O155	C102	P25	1000	21-4-78
O170	C111	P50	51	27-1-78
O257	C120	P37	600	11-1-78
O329	C102	P42	20	12-5-78

PRODOTTI

Codice	Descrizione	Prezzo
P25	Fanale	1.000
P37	Manubrio	10.500
P42	Bicicletta da corsa	200.000
P50	Bicicletta	120.000

Fig. 1

3. DESCRIZIONE DEI RAPPORTI

3.1 Selezione dalla base di dati

Un rapporto può essere costituito da dati provenienti da un sottoinsieme delle relazioni di base, eventualmente con un nome diverso da usare nell'intestazione e con gli attributi definiti in funzione di quelli originari.

Dalle relazioni di Fig. 1 è possibile, ad esempio, essere interessati alla relazione Clienti con un campo "area" definito in funzione della regione come in fig. 2.

Il formalismo per esprimere le funzioni sugli attributi e le condizioni dovrà essere scelto al momento dell'implementazione in funzione del sistema utilizzato. Negli esempi che seguono, verrà usata una sintassi che dovrebbe essere facilmente comprensibile da chi conosce un linguaggio programmativo tipo ALGOL.

3.2 Selezione da una relazione

Individuato l'insieme di relazioni di interesse al fine della produzione del rapporto, accade frequentemente di avere la necessità di stampare solo una

```
REPORT Clienti (codice, area, nome)
WHERE area IS CASE:
                regione = "piemonte" OR
                        = "lombardia"OR
                                •
                                •
                                •
                        THEN "settentrionale"
                regione = "toscana" OR
                        = "lazio" OR
                                •
                                •
                                •
                        THEN "centrale"
                regione = "campania" OR
                        = "calabria" OR
                                •
                                •
                                •
                        THEN "meridionale"
                regione = "sicilia" OR
                        = "sardegna"
                        THEN "insulare"
```

Fig. 2

parte dei dati che vi son contenuti. A tal fine, nel linguaggio è possibile specificare una condizione che permette di selezionare un sottoinsieme di tuple da una relazione.

Ad esempio, per la base di dati di Fig. 1 si può richiedere un rapporto sulla relazione Prodotti in cui compaiono solo alcuni domini delle tuple con prezzo minore di 100000:

```
REPORT Prodotti-a-buon-mercato (codice-prodotto, descrizione, prezzo)
       FROM Prodotti
       WHERE Codice-prodotto IS Codice
       WITH Prezzo < 100000
```

Una caratteristica molto importante dei sistemi per la generazione di rapporti è data dalla flessibilità concessa per definire la sequenza in cui i dati devono apparire.

3.3 Ordinamento

Una richiesta molto comune per la stampa di dati da una relazione è che siano ordinati in modo ascendente o discendente, rispetto ai valori di uno o più attributi.

3.4 Raggruppamento

Per la presentazione di una relazione può essere utile evidenziare gruppi di registrazioni con valori uguali di uno o più attributi. Un esempio di raggruppamento è dato dalla necessità di produrre in uscita le tuple della relazione Clienti raggruppate per regione come in Fig. 3.

Codice	Regione	Nome	Indirizzo
C101	Lazio	Albini	V. Nomentana Roma
C102	Lombardia	Zanardi	•
C120	Lombardia	•	•
C101	Toscana	Rossi	•
C110	Toscana	De Mita	•
C111	Toscana	•	•

Fig. 3

Questa operazione risulta particolarmente utile nel caso che si vogliano calcolare e stampare, per ogni gruppo, i valori di funzioni come il totale, il conteggio, il massimo, il minimo e la media. Un altro uso molto più significativo si vedrà nel caso di relazioni in corrispondenza.

Il concetto di raggruppamento si chiarisce riconducendolo a quello matematico di relazione di equivalenza su insiemi (°). Se A è un attributo di una relazione R, si può definire la relazione binaria $\equiv_A$ come relazione di equivalenza su R, con la proprietà che per due tuple t_1, t_2 e R si ha che $t_1 \equiv_A t_2$ se e solo se $t_1 [A] = t_2 [A]$, cioè se hanno lo stesso valore dell'attributo A.

La definizione si estende facilmente al caso in cui A sia un insieme di attributi. Il significato di questa operazione è quello di partizionare R in classi di equivalenza, cioè sottoinsiemi di tuple con uguale valore degli attributi in A.

La richiesta di raggruppare le tuple di una relazione in base ad un insieme di attributi implica anche un ordinamento dei gruppi rispetto ai valori degli attributi di raggruppamento. L'ordinamento, se non specificato, si assume ascendente.

L'uso combinato delle specifiche di raggruppamento e ordinamento produce l'effetto dell'ordinamento all'interno di un gruppo. Naturalmente

(°) Questo concetto è stato anche introdotto [4] per estendere la potenza dell'algebra relazionale.

nel caso di un'unica relazione la stessa sequenza può essere ottenuta con il solo ordinamento.

3.5 Corrispondenze tra due relazioni

Nella descrizione di un rapporto si ha la possibilità di richiedere di intercalare dati appartenenti a relazioni diverse quando fra esse esiste una corrispondenza.

Fra gli elementi di due relazioni esiste un corrispondenza diretta quando una relazione contiene un dominio (chiave esterna) che appare nell'altra come chiave primaria. Una corrispondenza diretta associa le relazioni nei due sensi facendo corrispondere ad una tupla di una relazione, con un certo valore della chiave primaria, tutte quelle nell'altra con lo stesso valore della chiave esterna e viceversa.

Sull'insieme delle relazioni si definisce una relazione-matematica binaria ϱ tale che R ϱ S quando esiste una corrispondenza diretta tra gli elementi di R e quelli di S. In questo contesto, due relazioni P e Q sono in corrispondenza o quando esiste una corrispondenza diretta oppure quando la coppia (P, Q) appartiene alla chiusura transitiva di ϱ.

Ad esempio, sull'insieme delle relazioni Clienti, Ordini e Prodotti esistono due corrispondenze dirette, una fra Clienti e Ordini e l'altra fra Ordini e Prodotti e una corrispondenza fra Clienti e Prodotti. In altre parole, è possibile mettere in corrispondenza le tuple dei clienti con i prodotti da essi ordinati e produrre un rapporto come se fra le relazioni esistesse una corrispondenza diretta.

3.6 Più relazioni in corrispondenza

Se si vogliono stampare più relazioni in corrispondenza fra loro, il linguaggio richiede che la struttura risultante sia un albero in cui cioè ogni relazione tranne la radice è codominio di una sola corrispondenza. La definizione dell'albero è necessaria perchè induce un ordinamento tra le relazioni che viene riprodotto nella stampa del rapporto. Infatti se R ed S sono due relazioni in corrispondenza ed R precede S nell'ordinamento sequenziale gerarchico, ogni tupla di R precede tutte le tuple di S ad essa associate. Ad esempio, siano R ed S rispettivamente i Clienti e gli Ordini; nel rapporto ogni cliente precede tutti i suoi ordini.

L'uso delle specifiche di raggruppamento in relazioni collegate da corrispondenze con una struttura gerarchica influisce sull'ordine con cui le registrazioni vengono presentate nel rapporto.

Consideriamo dapprima il caso semplice di un albero costituito da due sole relazioni R ed S e supponiamo che R, radice dell'albero, sia raggruppata su un insieme A di attributi. Allora tutte le tuple di R con lo stesso valore di A precedono tutte le tuple di S ad essa associate. Inoltre la relazione S può, a sua volta, essere raggruppata su un insieme di attributi propri B oppure sullo stesso insieme di attributi A di R.

Si prendano ad esempio come relazioni R ed S rispettivamente i Clienti e gli Ordini e A coincidente con "regione" e B con "data". Se la relazione Ordini è raggruppata su "regione" si ha la sequenza di Fig. 4 se invece è raggruppata su "data" si ha la sequenza di Fig. 5.

L'estensione di queste considerazioni ad una generica struttura ad albero è immediata e mostreremo un esempio nel prossimo paragrafo dopo la definizione del linguaggio per la descrizione del rapporto.

4. SINTASSI DEL FORMALISMO

La notazione usata per la definizione della sintassi è descritta da Wirth [11], ed è una variante della classica BNF. I simboli terminali sono racchiusi tra apici (se gli apici appaiono come simboli terminali sono scritti due volte). La ripetizione è denotata da parentesi graffe: { a } sta per ε | a | aa | aaa ..., con ε stringa vuota. L'opzionalità è espressa dalle parentesi quadre: [a] sta per a | ε. Le parentesi tonde servono per raggruppare: (a | b) c equivale a ac | bc. Tutte le produzioni della grammatica sono terminate da un punto.

La definizione della sintassi è in fig. 6.

La dichiarazione delle corrispondenze viene data con la specifica di MATCH dove ogni coppia stabilisce una corrispondenza diretta.

5. ESEMPI

Vediamo alcuni tipi di rapporti generabili dal seguente insieme di relazioni:

CLIENTI	Codice	Regione	Nome	Indirizzo
	C100	Lazio	Albini	•
	C102	Lombardia	Zanardi	•
	C120	Lombardia	Bossi	•

ORDINI	Codice	Codice-cliente	Codice-prodotto	•	•
	O091	C120	P50	•	•
	O125	C102	P37	•	•
	•	•	•	•	•

CLIENTI	Codice	Regione	Nome	Indirizzo
	C101	Toscana	Rossi	•
	C110	Toscana	De Mita	•
	C111	Toscana	Rossi	•

ORDINI	Codice	Codice-cliente	Codice-prodotto	•	•
	O055	C101	•	•	•
	O059	C111	•	•	•
	•	•	•	•	•
	•	•	•	•	•

Fig. 4

CLIENTI	Codice	Regione	Nome	Indirizzo
	C100	Lazio	Albini	•

CLIENTI	Codice	Regione	Nome	Indirizzo
	C102	Lombardia	Zanardi	•
	C120	Lombardia	Bossi	•

ORDINI	Codice	Codice-cliente	Codice-prodotto	Q.tà	Data
	O091	C120	P50	45	21-4-78
	O155	C102	P25	1000	21-4-78
	O120	C102	P37	500	11-1-78
	O057	C120	P37	600	11-1-78
	0329	C102	P42	20	12-5-78

CLIENTI	Codice	Regione	Nome	Indirizzo
	C101	Toscana	Rossi	•
	C110	Toscana	De Mita	•
	C111	Toscana	Rossi	•

ORDINI	Codice	Codice-cliente	Codice-prodotto	Q.tà	Data
	O055	C101	P37	1000	8-3-78
	O103	C110	P42	15	8-3-78
	O059	C111	P42	45	19-2-78
	O170	C111	P50	51	27-1-78

Fig. 5

```
prospetto = "REPORT" albero.

albero = relazione [ "("albero { "," albero } ")" ].

relazione = nome [ "("nome-attributo { "," nome-attributo } ")" ]
            [ "FROM" nome-relazione ]
            [ "WHERE" nome "IS" espressione
                        { "AND" nome "IS" espressione } ]
            [ "MATCH" coppia { "AND" coppia } ]
            [ "WITH" condizione ]
            [ "GROUPED BY" "("attrord { "," attrord } ")" ]
            [ ("AVG" | "SUM" | "COUNT" | "MIN" | "MAX") nome-attributo
                                                "." nome-relazione ]
            [ ""ORD" "("attrord { "," attrord } ")" ].

attrord = nome-attributo "." nome-relazione [ ("ASC" | "DISC") ].

attributo = nome-attributo "." nome-relazione "IS" espressione.

coppia = "("nome-attributo "." nome-relazione ","
           nome-attributo "." nome-relazione")".
```

Fig. 6

Aree [nome, descrizione]
Clienti [codice, area, nome indirizzo]
Ordini [codice, codice-cliente, codice-prodotto, quantità, data]
Prodotto [codice, descrizione]
Fornitori [codice, area, indirizzo]
Forniture [codice, codice-fornitore, descrizione]

Notare, negli esempi che seguono, come la possibilità di avere gli attributi di ordinamento, raggruppamento e quelli relativi alle corrispondenze non coincidenti renda il formalismo molto flessibile.

• Stampare le aree raggruppate per nome e ordinate in senso ascendente seguite dai relativi clienti, con codice minore di 200, raggruppati per area e ordinati sul codice (in senso ascendente), seguiti dagli ordini con data compresa tra 1-1-80 e 31-12-78 raggruppati per data e ordinati per quantità (in senso ascendente), seguiti ancora dai relativi prodotti. Inoltre, stampare i fornitori raggruppati per area seguiti dalle relative forniture (fig. 7).

Il risultato appare nella forma mostrata in fig. 8.

```
REPORT (aree
        GROUPED BY (nome . aree ASC)
       (clienti
        MATCH (nome . aree, area . clienti)
        WITH codice < 200
        GROUPED BY (nome . aree)
        ORD (nome . clienti ASC, codice . clienti ASC)
       (ordini-del-79
        FROM ordini
        MATCH (codice . clienti, codice-cliente . ordini)
            AND (codice . prodotti, codice-prodotto
                 . ordini)
        WITH 31-12-78 < data < 1-1-80
        GROUPED BY (data . ordini ASC)
        ORD (quantità . ordini ASC)
       (prodotti
        GROUPED BY (data)
        ORD (codice . prodotti)          )))
       (fornitori
        MATCH (nome . aree, area . fornitori)
        GROUPED BY (nome . aree ASC)
        ORD (codice . fornitori ASC)
       (forniture
        MATCH (codice . fornitori, codice-fornitore
                . forniture)
        GROUPED BY (nome . aree)
        ORD (codice. forniture)        )))
```

Fig. 7

AREE	Nome	Descrizione			
	Centrale	• • •			
CLIENTI	Codice	Regione	Nome	Indirizzo	
	C100	Centrale	Albini	•	
	C101	Centrale	Rossi	•	
	C111	Centrale	Rossi	•	
ORDINI	Codice	Codice-cliente	Codice prodotto	Q.tà	Data
	O245	C100	P42	1	12-1-79
	O390	C111	P50	4	12-1-79
PRODOTTI	Codice	Descrizione	Prezzo		
	P42	Bicicl. corsa	200.000		
	P50	Bicicletta	120.000		
ORDINI	Codice	Codice-cliente	Codice-prodotto	Q.tà	Data
	O459	C101	P50	6	15-5-79
	O555	C111	P37	10	15-5-79
	O586	C100	P25	16	15-5-79
PRODOTTI	Codice	Descrizione	Prezzo		
	P25	Fanale	1.000		
	P37	Manubrio	10.500		
	P50	Bicicletta	120.000		
FORNITORI	Codice	Area	Indirizzo		
	FO18	Centrale	•		
	FO25	Centrale	•		
FORNITURE	Codice	Codice-fornitore	Descrizione		
	FU45	FO25	•		
	FU78	FO18	•		
	FU93	FO18	•		
AREE	Nome	Descrizione			
	•	•			
	•				
	•				
	•				

Fig. 8

• Stampare i clienti dell'Italia settentrionale, ordinati per nome e codice (entrambi in senso ascendente) e i relativi prodotti (fig. 9).

Il risultato appare nella forma mostrata in fig. 10.

• Stampare gli ordini del 79 raggruppati per codice-prodotto (in senso ascendente), ordinati per codice (in senso ascendente) e stampare la somma delle quantità ordinate di ogni prodotto (fig. 11). Il risultato appare nella forma mostrata in fig. 12.

```
REPORT ( clienti
         WITH area . clienti = "settentrionale"
         ORD (nome . clienti = ASC, codice . clienti ASC)
       ( prodotti
         MATCH (codice . clienti, codice . cliente . ordini)
               AND (codice . prodotti, codice-prodotto . ordini)
         ORD (codice . prodotti ASC)          ))
```

Fig. 9

CLIENTI	Codice	Area	Nome	Indirizzo
	C120	Settentrionale	Bossi	
PRODOTTI	Codice	Descrizione	Prezzo	
	P37	Manubrio	10.555	
	P50	Bicicletta	120.000	
CLIENTI	Codice	Area	Nome	Indirizzo
		Settentr.	Zanardi	•
PRODOTTI	Codice	Descrizione	Prezzo	
	P25	Fanale	1.000	
	P37	Manubrio	•	
	P42	Bicicletta corsa	•	

Fig. 10

```
REPORT ( ordini (codice, codice-prodotto, quantità, data)
         WITH 31-12-78 < data < 1-1-80
         GROUPED BY (codice-prodotto . ordini ASC)
         SUM (quantità . ordini)    )
```

Fig. 11

ORDINE	Codice	Codice-prodotto	Quantità	Data
	O121	P25	125	3-1-79
	O145	P25	38	13-5-79
	O171	P25	100	11-4-79
			SUM (quantità . ordini) = 263	
	O115	P37	50	10-2-79
	O221	P37	30	20-6-79
			SUM (quantità . ordini) = 80	
	•	•	•	
	•	•	•	

Fig. 12

6. CONCLUSIONI

È stato descritto un formalismo per dichiarare rapporti costituiti da dati appartenenti a insiemi diversi. La presentazione è stata orientata verso un modello dei dati relazionale, ma il formalismo è facilmente adattabile ad ogni altro modello basato sui concetti di entità, proprietà e associazione.
In questa versione del lavoro, l'accento è stato posto principalmente sulle possibilità offerte all'utente di organizzare logicamente i dati nel rapporto, riducendo al minimo ogni considerazione sulle modalità di visualizzazione degli stessi e sull'aggiunta di righe di intestazione. Questo problema, insieme a quello della realizzazione e alla possibilità di uso di supporti grafici interattivi, andrà affrontato una volta fissato il sistema nel quale dovrà essere utilizzato il formalismo proposto e in funzione del tipo di linguaggio disponibile per operare sulla base di dati.

7. BIBLIOGRAFIA

[1] Bracchi, G., Martella, G. e Pelagatti, G., SISTEMI PER LA GESTIONE DI BASI DI DATI, ISEDI, Milano, 1979

[2] Date, C.J., AN INTRODUCTION TO DATA BASE SYSTEMS, Addison Wesley, Reading, Mass. (2ª edizione), 1977

[3] Fry, J.P. and Sibley, E.H., EVOLUTION OF DATA BASE MANAGEMENT SYSTEMS, Computing Surveys 8, 1, 1976

[4] Furtado, A.L. and Kerschberg, L., AN ALGEBRA OF QUOTIENT RELATION, ACM Sigmod, 1977

[5] Kim, W., RELATIONAL DATA BASE SYSTEMS, Computing Surveys 11, 3, 1979

[6] Morelli, M., INFORMATICA, Angeli, Milano, 1976

[7] Postley, J.A., THE MARK IV SYSTEM, Datamation 14, 1, 1978

[8] Wirth, N., WHAT CAN WE DO ABOUT THE UNNECESSARY DIVERSITY ON NOTATION FOR SYNTACTIC DEFINITIONS?, CAMC 20, 11, 1977

UN SISTEMA AD INDICE A DUE LIVELLI

Le Van Huu
Istituto di Cibernetica - Università di Milano

Riassunto

L'obiettivo di questo articolo è di descrivere un sistema automatico di indicizzazione, chiamato KWOC, utile per la catalogazione di documenti.
Presentiamo, di tale sistema, la versione originale e una versione estesa, insieme ad alcune proposte per l'applicazione pratica, in vari ambienti, del sistema.
Infine, l'articolo conclude con un suggerimento per determinare la distribuzione delle parole chiave che sono estratte automaticamente dai titoli di documenti da catalogare.

Abstract

The aim of the present paper is to describe an automatic indexing system, called KWOC, suitable to catalogue documents.
We present of such a system its basic version and an extended version as well as a few hints for its practical use in various environments.
Finally the paper concludes with a suggestion to determinate the distribution of keywords which are automatically sorted out from titles of documents to be catalogued.

1. INTRODUZIONE

L'introduzione dei calcolatori nei sistemi di Information Retrieval ha avuto fra le prime applicazioni la produzione di indici, intesi come liste stampate secondo determinati formati, contenenti informazioni utili per il reperimento di oggetti.

Fra i più conosciuti sistemi automatizzati di indicizzazione vi è il Kwoc, (Key-Words Out of Context), utile per l'ordinamento di documenti e molto usato nelle biblioteche e negli ambienti ove occorre la gestione di una quantità ingente di documenti.

Questo articolo darà una visione generale relativa alla struttura, all'organizzazione e all'uso del sistema Kwoc. Saranno presentate, inoltre, l'estensione di tale sistema, effettuata dall'Istituto di Cibernetica dell'Università di Milano, e alcune proposte per la sua applicazione in diversi ambienti. Infine, verrà riportato uno studio per determinare la distribuzione delle parole significative estratte dai titoli di documenti da ordinare. Queste parole sono presenti in uno degli indici prodotti dal sistema Kwoc. La conoscenza di questa distribuzione può essere utile per definire l'importanza e il peso delle parole considerate significative all'interno dell'indice in esame.

2. IL SISTEMA KWOC

È stato realizzato intorno alla fine degli anni 60 dall'AMES Laboratory USAEC dell'Università di IOWA (U.S.A) ed è un'estensione del sistema Kwic (Key-Words In Context). La funzione del sistema Kwoc è quella di esaminare l'insieme delle informazioni dei documenti (la precisazione verrà data fra breve) fornitigli come dati di ingresso e, quindi, di produrre degli indici contenenti queste informazioni, ordinate secondo determinati formati.

Intendiamo con "documenti" un insieme di oggetti omogenei (per esempio libri, lettere e simili) da trattare con il sistema Kwoc. Ogni documento è caratterizzato da vari tipi di informazioni, chiamati **descrittori**.

I descrittori necessari per il funzionamento del Kwoc sono:

- titolo (100 caratteri alfanumerici)
- autore/i (possono essere presenti 27 nomi di autori, ciascuno di 20 caratteri alfanumerici)
- codice base, o codice d'archivio (di 11 caratteri), unico per ogni documento ed assegnato dai responsabili del sistema (°) secondo determinati criteri.

Altri descrittori, quali:

- note bibliografiche (casa editrice, anno pubblicazione . . .)
- riassunti

sono facoltativi.

(°) Nel seguito chiameremo **utenti** le persone che consultano gli indici Kwoc, mentre con **responsabili del sistema** intendiamo le persone che si occupano del sistema stesso in tutti suoi aspetti, dalla preparazione dei dati di ingresso all'esecuzione dei programmi, e così via.

Il sistema Kwoc permette, in uno stesso ciclo di esecuzione, il trattamento di dati di ingresso costituiti da documenti di natura diversa fra loro; in tal caso è necessario il descrittore **intestazione** per la distinzione dei documenti.

Vedremo in seguito come i nomi dei descrittori possano essere usati nelle varie applicazioni in un senso anche molto lato, allontanandosi dall'ambiente proprio di una biblioteca.

Dall'elaborazione dei descrittori, vengono prodotti i seguenti indici:

- **indice base**, ordinato alfabeticamente secondo i codici base, contenente per ogni documento:
  - nomi degli autori
  - titolo

  ed eventualmente
  - note bibliografiche
  - riassunti

- **indice per autore**, contenente i nomi degli autori ordinati alfabeticamente e i codici base dei relativi documenti

- **indice per parole chiave**, contenente in ordine alfabetico tutte le **parole chiave** estratte dai descrittori titolo, accompagnate dai titoli dei documenti da cui sono state estratte e dai loro codici base.

Con il termine parole chiave ci riferiamo alle parole contenute nel titolo del documento e ritenute significative per l'individuazione del documento stesso.
Le parole chiave vengono automaticamente scelte per confronto fra le parole dei titoli e una tabella contenente le parole ritenute a priori poco significative dai responsabili del sistema.
Ovviamente, i titoli che contengono più di una parola chiave compaiono in più posti dell'indice.

3. ESTENSIONE DEL KWOC

Allo scopo di migliorare ed estendere le prestazioni del sistema, intorno al 1977, sono state apportate al Kwoc, da parte dell'Istituto di Cibernetica, le seguenti modifiche (effettuate sull'elaboratore Honeywell Level 62).

La consultazione dell'indice per parole chiave prodotto dal Kwoc originario comporta in alcuni casi un lavoro di spoglio molto lungo. Ovvero, quando più titoli hanno in comune una parola chiave, vengono stampati nell'indice accanto alla parola stessa. L'utente è costretto quindi a leggere questi titoli prima di poter decidere quali di essi siano attinenti alla sua ricerca.

Questo inconveniente è attenuato dal sistema Kwoc esteso, il quale mette in rilievo nell'indice, oltre alla parola chiave in comune, anche una seconda parola estratta dai titoli.
In questo modo l'utente ha la possibilità, una volta individuata la parola chiave principale, di ridurre la quantità di documenti pertinenti scegliendo fra le seconde parole chiave quelle di maggior interesse.

Nella scelta di quale parola mettere in rilievo come seconda parola chiave si è però aperta subito almeno una alternativa: privilegiare la parola immediatamente precedente o quella immediatamente successiva. Una scelta ragionata richiederebbe di tenere in considerazione almeno i seguenti elementi (e le loro interrelazioni): posizione nel titolo della prima parola chiave, lingua in cui è formulato il titolo, categoria grammaticale della prima parola chiave, categoria grammaticale della eventuale seconda, distanza fra le due nel titolo, etc. Si è perciò preferito operare una scelta a priori, e precisamente a favore della parola chiave immediatamente successiva, quando vi sia, obbedendo piuttosto al criterio di "avanzamento del testo".

Una seconda estensione riguarda la possibilità di estrarre parole chiave soltanto da una parte dell'intero titolo. A questo proposito le parti del titolo ritenute interessanti per l'estrazione delle parole chiave sono state racchiuse fra due caratteri speciali. Ad esse è stato assegnato il nome di "frasi chiave".

4. APPLICAZIONI DEL KWOC IN AMBIENTI DIVERSI

Date le caratteristiche descritte, il sistema Kwoc può venire introdotto, oltre che nelle biblioteche in senso stretto, in tutti gli ambienti in cui deve essere gestita una quantità ingente di documenti.
Consideriamo qui i seguenti tipi di documenti e ambienti relativi:

- articoli scientifici
- delibere

- corrispondenza
- questionari
- teoremi matematici

4.1 Articoli scientifici

In una biblioteca contenente un numero elevato di riviste, la ricerca di un articolo relativo ad un determinato argomento può comportare un enorme lavoro manuale di spoglio e molto tempo di consultazione e lettura. Per ridurre tempi e fatica, ogni articolo viene classificato come un documento distinto, con descrittori distribuiti nel modo seguente:

- **Titolo** è considerato il titolo dell'articolo seguito dal suo riassunto, se c'è.

- Per ogni articolo, il **codice base** (11 caratteri) contiene le seguenti informazioni:
  - sigla della rivista (4 caratteri)
  - anno di pubblicazione (2 caratteri)
  - numero progressivo della rivista (3 caratteri)
  - numero d'ordine dell'articolo all'interno del numero considerato.

Per esempio, il codice base NOSW7900501 si riferisce al primo articolo della rivista "Note di Software", numero 5, anno 1979.

- **Autori** sono gli autori dell'articolo.

- altri descrittori non vengono usati.

Un esempio dell'indice base e dell'indice per autori si trova nella figura 1.

I vantaggi degli indici prodotti da questa applicazione non sono pochi. La ricerca per argomento può avvenire sull'indice per parole chiave: il titolo ed eventualmente il riassunto dell'articolo affiancati alle parole chiave aiutano nella valutazione della pertinenza dell'articolo stesso. Inoltre, individuati gli articoli di interesse, i relativi codici base permettono di reperire le riviste desiderate.

D'altra parte, l'indice base, ordinato secondo le sigle delle riviste, offre informazioni (titolo, riassunto, autori . . .) sulla struttura e la storia delle varie riviste. Una lista in cui siano associati alle sigle adottate (soltanto 4 caratteri) i nomi effettivi delle riviste completerà le informazioni.

4.2 Delibere

Per "delibera" si intende un documento che riporta le decisioni di Enti pubblici, quali Giunta comunale, Assessorati, etc., su questioni di ordine pubblico.

Ogni delibera viene considerata come un singolo documento con la seguente distribuzione di informazioni:

- nel **codice base**:
  - la data di approvazione della delibera (6 caratteri)
  - un carattere vuoto
  - il numero progressivo della delibera (4 caratteri)

- nel **titolo**, l'oggetto della delibera

- nell'**autore**, informazioni di vario genere, e precisamente:
  - l'organo che ha approvato la delibera
  - la materia della delibera
  - il nome dell'Ente

Per una chiara distinzione fra tali elementi e per aumentare la potenzialità dell'indice per autori, sono state premesse le seguenti lettere:

A, davanti al nome dell'organo deliberante
B, davanti alla materia
C, davanti al nome dell'Ente

Per esempio, l'Assessorato all'edilizia privata apparirà come autore in questo formato:

A EDILIZIA PRIVATA

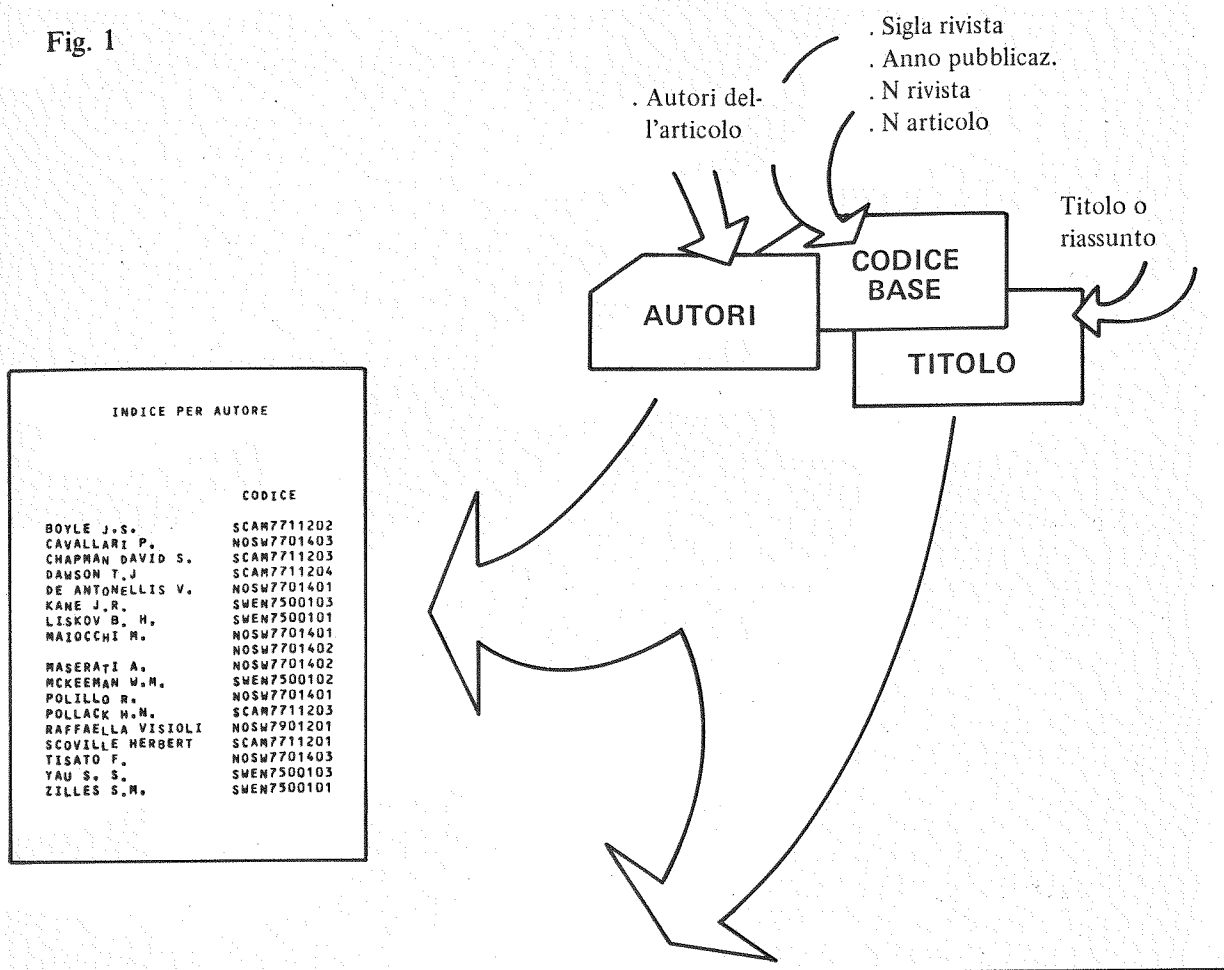
oppure una delibera relativa al piano regolatore avrà, sotto il descrittore "autore":

B PIANO REGOLATORE

Quando l'argomento della delibera sia bene espresso da parole isolate, la ricerca potrà avvenire sull'indice per parole chiave. Le date indicate nei codici base delle delibere trovate attraverso questa ricerca consentono una buona selezione delle delibere stesse.

In questa applicazione l'indice per autore riveste una particolare importanza. Esso permette infatti di svolgere una ricerca anche quando si conosca soltanto uno dei tre tipi di "autori" considerati. In altre parole, quando si vogliano individuare delibere approvate da un Assesorato, si cerca il nome di quell'Assesorato nel tipo che inizia con la lettera A; analogamente, la ricerca delle delibere relative a una determinata materia viene effettuata con la lettera B iniziale; e così via.

Fig. 1



INDICE PER AUTORE

	CODICE
BOYLE J.S.	SCAM7711202
CAVALLARI P.	NOSW7701403
CHAPMAN DAVID S.	SCAM7711203
DAWSON T.J	SCAM7711204
DE ANTONELLIS V.	NOSW7701401
KANE J.R.	SWEN7500103
LISKOV B. H.	SWEN7500101
MAIOCCHI M.	NOSW7701401
	NOSW7701402
MASERATI A.	NOSW7701402
MCKEEMAN W.M.	SWEN7500102
POLILLO R.	NOSW7701401
POLLACK H.N.	SCAM7711203
RAFFAELLA VISIOLI	NOSW7901201
SCOVILLE HERBERT	SCAM7711201
TISATO F.	NOSW7701403
YAU S. S.	SWEN7500103
ZILLES S.M.	SWEN7500101

CODICE INDICE BASE UNIVERSITA' DI MILANO

NOSW7701401 DE ANTONELLIS V. MAIOCCHI M. POLILLO R.
SCHEMI DI DIALOGO : UNO STRUMENTO PER LA RAPPRESENTAZIONE DEI DIALOGHI IN UN SISTEMA INTERATTIVO.

NOSW7701402 MAIOCCHI M. MASERATI A.
PHOS ; UNA METODOLOGIA PER LA COSTRUZIONE VELOCE ED AFFIDABILE DI PROGRAMMI.

NOSW7701403 CAVALLARI P. TISATO F.
IL SIMULA 67 COME LINGUAGGIO DI SPECIFICA ESEGUIBILE

NOSW7901201 RAFFAELLA VISIOLI
LIBRO-BIANCO : UNO STRUMENTO PER LA GESTIONE DEI TESTI

SCAM7711201 SCOVILLE HERBERT
I NEGOZIATI SALT

SCAM7711202 BOYLE J.S.
COMUNICAZIONE SU ONDA LUMINOSA

SCAM7711203 POLLACK H.N. CHAPMAN DAVID S.
IL FLUSSO DI CALORE DALL'INTERNO DELLA TERRA

SCAM7711204 DAWSON T.J
I CANGURI

SWEN7500101 LISKOV B. H. ZILLES S.M.
SPECIFICATION TECHNIQUES FOR DATA ABSTRACTIONS

SWEN7500102 MCKEEMAN W.M.
ON PREVENTING PROGRAMMING LANGUAGES FROM INTERFERING WITH PROGRAMMING

SWEN7500103 KANE J.R. YAU S. S.
CONCURRENT SOFTWARE FAULT DETECTION

NOSW7701401 * SWEN7500103

È possibile anche una ricerca incrociata fra più tipi. Per esempio, per una delibera che riguardi una certa materia e che sia stata approvata da un determinato Assesorato, si selezionano prima i codici base relativi alla materia cercata, poi quelli corrispondenti all'Assesorato in esame. Fra i codici base comuni ai due gruppi selezionati si troverà la delibera o le delibere desiderate.

Un esempio degli indici relativi ad alcune delibere è riportato in figura 2.

4.3 Corrispondenza

Proponiamo due possibili tipi di applicazione di Kwoc alla gestione della corrispondenza.

Il primo mette in rilievo due aspetti di importanza non trascurabile della gestione della corrispondenza. Innanzitutto, una **lettera** può assumere significati diversi a seconda che sia considerata di proprietà dei destinatari oppure dei mittenti. Inoltre, succede spesso che il numero dei destinatari, o dei mittenti, di una lettera sia più di uno.

A questo scopo, il sistema Kwoc considera ogni lettera non come un singolo documento, ma come un insieme di documenti, quanti sono i suoi destinatari e i suoi mittenti. Una lettera inviata dal mittente A ai destinatari B e C sarà considerata come l'insieme di tre documenti distinti, appartenenti rispettivamente al gruppo dei mittenti e al gruppo dei destinatari, e precisamente:

- il primo documento appartiene al gruppo delle lettere inviate e si riferisce al mittente A;
- il secondo e il terzo appartengono al gruppo delle lettere relative ai destinatari B e C.

Per ogni documento occorre indicare:

- nel descrittore **titolo** l'oggetto della lettera oppure un riassunto del suo contenuto. Si può ricorrere anche al descrittore facoltativo **bibliografie** per indicare tutti i nomi dei mittenti e dei destinatari. Queste informazioni verranno stampate nell'indice base, dopo il titolo di ogni documento.

- Nel descrittore **autore**, il cognome e il nome del proprietario della lettera.

- Nel **codice base** le seguenti informazioni:
  - nel primo carattere la lettera M se il documento è considerato come una lettera inviata, altrimenti la lettera D. Questo accorgimento ha lo scopo di evidenziare nell'indice base, ordinato secondo i codici base, le lettere inviate e quelle ricevute
  - l'iniziale del cognome, oppure una sigla, del mittente/destinatario
  - l'anno e il mese in cui è stata inviata la lettera
  - il numero progressivo della lettera all'interno del mese.

Un esempio del codice base per la corrispondenza può essere:

DROS7902124

per la lettera che è stata ricevuta dal destinatario con la sigla ROS e che è stata scritta nel febbraio del 1979, come 124-esima lettera del mese, dalle persone che fanno parte dell'organizzazione.

La figura 3 rappresenta l'indice base e l'indice per autore relativi alle seguenti due lettere:

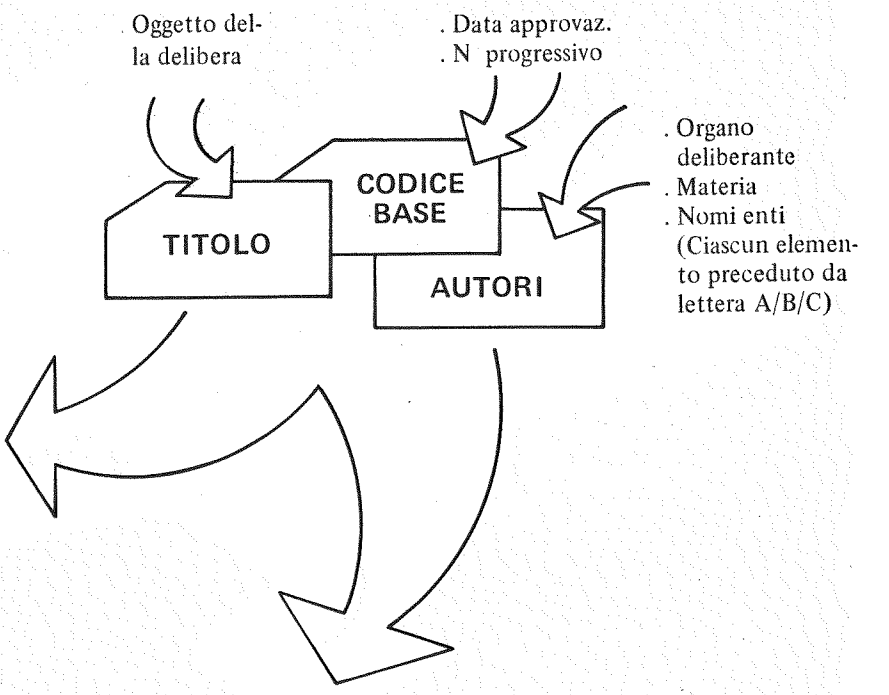
1) inviata da ROSSI M. a BIANCHI L. e VERDI F. nel giorno 11/2/78 a proposito di una riunione;
2) inviata da BIANCHI L. a VERDI F. nel giorno 20/2/78 per informazioni relative al corso di Cibernetica.

Attraverso l'indice per autore, l'utente può individuare i codici base delle lettere che gli si riferiscono, e, mediante le iniziali, M o D, distinguere le lettere inviate da quelle ricevute, con una selezione ancora più restrittiva se gli è nota anche la data della lettera cercata. Analogamente, consultando l'indice base, l'utente può individuare le lettere ricevute e quelle inviate e, una volta trovati i codici che contengono le lettere iniziali del suo cognome, valutare gli oggetti e le note delle lettere stampate accanto.

Se, tuttavia, la moltiplicazione dei documenti, dovuta alla trasformazione di ciascuna lettera in più documenti, fosse ritenuta per qualche motivo eccessiva, si può ricorrere ad una semplificazione.
Il secondo tipo di applicazione, infatti, considera ogni lettera come un singolo documento. Si indicano allora nel descrittore **autore** tutti i cognomi dei mittenti e dei destinatari, preceduti rispettivamente dalla lettera M oppure D, e nel codice base le seguenti informazioni:

- la data d'invio della lettera (6 caratteri)
- il numero progressivo della lettera (5 caratteri).

Fig. 2



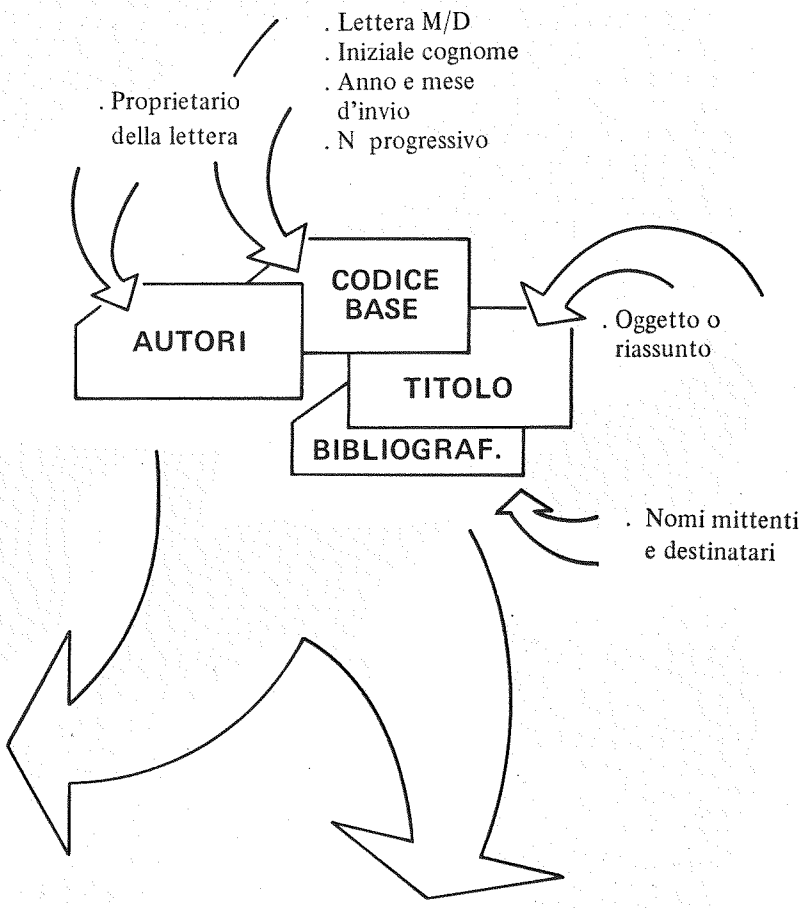
INDICE PER AUTORE

		CODICE
A	EDILIZIA PUBBLICA	770315 1322
A	PUBBL.ISTRUZIONE	750910 1732
B	ASSISTENZA	770107 1123
B	CONSORZIO RIFIUTI	751210 0234
B	FOGNATURE	770315 1322
B	PERSONALE	770310 1321
B	PIANO REGOLATORE	761130 2341
B	STRADE	770315 1322
C	CASATENOVO	750910 1732
		751210 0234
		761102 1981
		761130 2341
		770107 1123
		770310 1321
		770315 1322

CODICE	INDICE CHIAVE	UNIVERSITA' DI MILANO PAGINA 1
750910 1732	ANNO-SCOL-75-76	LEGGE-REG-3-5-74 N-59 APPROVAZIONE PIANO COMUNALE DI INTERVENTO PER L' ANNO-SCOL-75-76
770107 1123	ANNO-75	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
770310 1321	APPROVAZIONE CONGEDO	APPROVAZIONE PER IL CONGEDO STRAORDINARIO PER MATERNITA ALLA IMPIEGATA DI RUOLO DEL COMUNE SIGNORA BIANCHI
750910 1732	PIANO	LEGGE-REG-3-5-74 N-59 APPROVAZIONE PIANO COMUNALE DI INTERVENTO PER L' ANNO-SCOL-75-76
770107 1123	ASSISTENZA	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
770107 1123	ASSOCIAZIONE	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
770310 1321	BIANCHI	APPROVAZIONE PER IL CONGEDO STRAORDINARIO PER MATERNITA ALLA IMPIEGATA DI RUOLO DEL COMUNE SIGNORA BIANCHI
770315 1322	CASCINA-CROTA	PERIZIA-SUPPLETIVA DEI LAVORI DI COSTRUZIONE DELLA STRADA E FOGNATURE PER LE SCUOLE DI VIA SAN-GIACOMO A CASCINA-CROTA
750910 1732	COMUNALE COMUNE	LEGGE-REG-3-5-74 N-59 APPROVAZIONE PIANO COMUNALE DI INTERVENTO PER L' ANNO-SCOL-75-76
770310 1321	SIGNORA	APPROVAZIONE PER IL CONGEDO STRAORDINARIO PER MATERNITA ALLA IMPIEGATA DI RUOLO DEL COMUNE SIGNORA BIANCHI
761130 2341	URBANISTICO	CONTRODEDUZIONE ALLE PROPOSTE REGIONALI DI MODIFICA DEL PIANO-REGOLATORE GENERALE URBANISTICO DEL COMUNE
761102 1981	COMUNICAZIONE	COMUNICAZIONE DELLA PRESIDENZA
770310 1321	CONGEDO	APPROVAZIONE PER IL CONGEDO STRAORDINARIO PER MATERNITA ALLA IMPIEGATA DI RUOLO DEL COMUNE SIGNORA BIANCHI
751210 0234	CONSORZIO	CONSORZIO INTERCOMUNALE PER LA ELIMINAZIONE DEI RIFIUTI-SOLIDI : NOMINA RAPPRESENTANTE EFFETTIVO E SUPPLENTE
770107 1123	CONTRIBUTI	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
761130 2341	CONTRODEDUZIONE	CONTRODEDUZIONE ALLE PROPOSTE REGIONALI DI MODIFICA DEL PIANO-REGOLATORE GENERALE URBANISTICO DEL COMUNE
770315 1322	COSTRUZIONE	PERIZIA-SUPPLETIVA DEI LAVORI DI COSTRUZIONE DELLA STRADA E FOGNATURE PER LE SCUOLE DI VIA SAN-GIACOMO A CASCINA-CROTA
770107 1123	EDUCAZIONE	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
751210 0234	EFFETTIVO	CONSORZIO INTERCOMUNALE PER LA ELIMINAZIONE DEI RIFIUTI-SOLIDI : NOMINA RAPPRESENTANTE EFFETTIVO E SUPPLENTE
751210 0234	ELIMINAZIONE	CONSORZIO INTERCOMUNALE PER LA ELIMINAZIONE DEI RIFIUTI-SOLIDI : NOMINA RAPPRESENTANTE EFFETTIVO E SUPPLENTE
770107 1123	ENTI	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
770107 1123	FAVORE	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
770315 1322	FOGNATURE	PERIZIA-SUPPLETIVA DEI LAVORI DI COSTRUZIONE DELLA STRADA E FOGNATURE PER LE SCUOLE DI VIA SAN-GIACOMO A CASCINA-CROTA
761130 2341	GENERALE	CONTRODEDUZIONE ALLE PROPOSTE REGIONALI DI MODIFICA DEL PIANO-REGOLATORE GENERALE URBANISTICO DEL COMUNE
770310 1321	IMPIEGATA	APPROVAZIONE PER IL CONGEDO STRAORDINARIO PER MATERNITA ALLA IMPIEGATA DI RUOLO DEL COMUNE SIGNORA BIANCHI
770107 1123	INFANZIA	CONTRIBUTI PER L'ANNO-75 IN FAVORE DEGLI ENTI ED ASSOCIAZIONE OPERANTI NEL SETTORE DELLA ASSISTENZA E DELLA EDUCAZIONE DELLA INFANZIA
751210 0234	INTERCOMUNALE	CONSORZIO INTERCOMUNALE PER LA ELIMINAZIONE DEI RIFIUTI-SOLIDI : NOMINA RAPPRESENTANTE EFFETTIVO E SUPPLENTE
750910 1732	INTERVENTO	LEGGE-REG-3-5-74 N-59 APPROVAZIONE PIANO COMUNALE DI INTERVENTO PER L' ANNO-SCOL-75-76
770315 1322	LAVORI	PERIZIA-SUPPLETIVA DEI LAVORI DI COSTRUZIONE DELLA STRADA E FOGNATURE PER LE SCUOLE DI VIA

ANNO-SCOL-75-76 * LAVORI

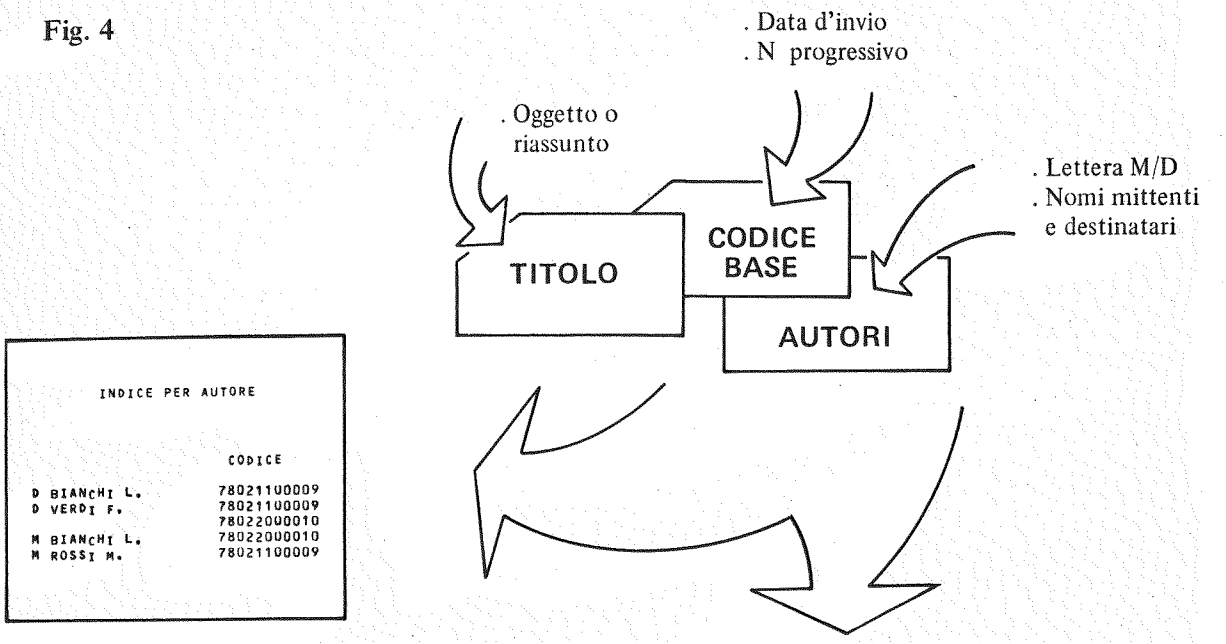
Fig. 3




```
INDICE PER AUTORE

                    CODICE

BIANCHI L.          DBIA7802009
                    MBIA7802010
ROSSI M.            MROS7802009
VERDI F.            DVER7802009
                    DVER78U2010
```

```
CODICE                          INDICE BASE          UNIVERSITA' DI MILANO    PAGINA   1

DBIA7802009   BIANCHI L.
              INVITO ALLA RIUNIONE PER IL BILANCIO
              MITTENTI : ROSSI M. - DESTINATARI : BIANCHI L. VERDI F.

DVER7802009   VERDI F.
              INVITO ALLA RIUNIONE PER IL BILANCIO
              MITTENTI : ROSSI M. - DESTINATARI : BIANCHI L. VERDI F.

DVER7802010   VERDI F.
              RICHIESTE DI INFORMAZIONI
              MITTENTI : BIANCHI L. - DESTINATARI : VERDI F.

MBIA7802010   BIANCHI L.
              RICHIESTE DI INFORMAZIONI
              MITTENTI : BIANCHI L. - DESTINATARI : VERDI F.

MROS7802009   ROSSI M.
              INVITO ALLA RIUNIONE PER IL BILANCIO
              MITTENTI : ROSSI M. - DESTINATARI : BIANCHI L. VERDI F.

                                    DBIA7802009 * MROS7802009
```

Fig. 4



INDICE PER AUTORE

	CODICE
D BIANCHI L.	78021100009
D VERDI F.	78021100009
	78022000010
M BIANCHI L.	78022000010
M ROSSI M.	78021100009

CODICE INDICE BASE UNIVERSITA' DI MILANO PAGINA 1

78021100009 D BIANCHI L. D VERDI F. M ROSSI M.
INVITO ALLA RIUNIONE PER IL BILANCIO

78022000010 D VERDI F. M BIANCHI L.
RICHIESTE DI INFORMAZIONI

78021100009 * 78022000010

Le lettere dell'esempio precedente in questa applicazione assumerebbero i formati riportati in figura 4.

Questo secondo tipo di applicazione, mentre presenta una certa semplicità nell'organizzazione dei dati di ingresso e evita l'inconveniente di trasformare ogni lettera in più documenti, non offre all'utente la possibilità di accedere direttamente all'indice base qualora desideri individuare una lettera di sua proprietà, in quanto i codici base (secondo cui è ordinato l'indice base) non contengono più le iniziali dei cognomi dei mittenti/destinatari come nella prima proposta.

A seconda quindi delle esigenze dell'ambiente in cui viene gestita la corrispondenza, l'utente può scegliere uno fra i due tipi di applicazione presentati.

4.4 Questionario

Un altro settore di applicazione del Kwoc è rappresentato dai questionari, ed in particolare dai questionari, o moduli, contenenti una serie di domande rivolte a più persone per stime statistiche relative a determinati argomenti. L'ambito, per di più, è ristretto ai questionari a risposta libera e non comprende quelli a risposte prefabbricate.

Considerando ogni risposta come un singolo documento, i documenti vengono distinti nel modo seguente:

- nel **codice base** si indicano:
  - il numero distintivo del questionario (3 caratteri)

- - la lettera D e il numero progressivo (2 caratteri) della domanda contenuta nel questionario al quale si riferisce il documento
- - la lettera R e il numero progressivo della risposta riferita alla domanda in esame.

Per esempio, il codice 123D02R0045 si riferisce alla risposta della 45-esima persona (R0045) alla seconda domanda (D02) del questionario 123.

- Come **titolo** si indica l'intera risposta.

Per ottenere una maggiore comprensibilità nell'indice base, si può introdurre un documento fittizio per ogni domanda del questionario.
Tale documento avrà il numero progressivo di risposta uguale a zero e nel titolo la domanda.

- Come **autore** si indica il nome della persona che ha compilato il questionario.

Mostriamo (fig. 5) un esempio di due questionari (123 e 124) e supponiamo che le risposte siano 4 per il questionario 123 e 3 per il 124.

Gli indici prodotti sono riportati in figura 6.

L'indice base è suddiviso in gruppi a seconda del numero del questionario. All'inizio di ogni gruppo si trova il documento fittizio contenente la domanda, seguito dai documenti risposta. Si ha così una visione di insieme di risultato di ogni questionario.

L'indice per parole chiave è particolarmente utile quando si voglia sapere se ad una certa domanda è stata data una determinata risposta. Per esempio, per sapere se il cinema è compreso fra le risposte alla domanda sul tempo libero del questionario 123, sarà proprio la parola "cinema" a venire ricercata nell'indice per parole chiave. Una volta trovata la parola, mediante il codice base stampato a fianco, si controllerà se il documento che la contiene si riferisce proprio al questionario 123.

4.5 Teoremi matematici

Ogni teorema è considerato come un singolo documento con la seguente distribuzione di informazioni:

- nel **codice base**:
  - - l'argomento del teorema eventualmente abbreviato (8 caratteri)
  - - il numero progressivo del teorema relativo a tale argomento

Così, il codice DERIVATE102 indica il 102° teorema sulle derivate.

- nel **titolo**:

  - - l'enunciato del teorema. Poiché è stato osservato che nella ricerca di un teorema per parole chiave vengono abitualmente usati i termini contenuti nella formulazione della tesi, e non quelli dell'ipotesi, soltanto le tesi sono considerate quali frasi chiave.

- nell'**autore**:

  - - la disciplina alla quale il teorema appartiene (per esempio, geometria, analisi, etc.).

Un esempio degli indici è contenuto in figura 7.

L'utente a cui sia noto l'argomento di un teorema selezionerà nell'indice base i codici contenenti la sigla dell'argomento voluto: lo guideranno nella selezione i descrittori **autore** e **titolo**.

Quando l'utente sia invece in grado di esprimersi meglio mediante le parole della tesi, sarà l'indice per parole chiave ad essergli utile. Attraverso i codici base stampati accanto alle parole chiave e contenenti le abbreviazioni degli argomenti, potrà individuare i teoremi che si riferiscono allo argomento desiderato. Leggendo gli enunciati dei teoremi selezionati, potrà infine estrarre facilmente il teorema cercato.

5. DISTRIBUZIONE DELLE PAROLE CHIAVE NELL'INDICE KWOC

Come si è accennato, i programmi del sistema Kwoc, per riconoscere le parole chiave nel descrittore titolo, ricorrono a una tabella contenente le parole ritenute a priori non significative, chiamate anche parole vietate. Ogni parola del titolo viene confrontata con quelle della tabella e l'esito negativo del confronto definisce la parola in esame come parola chiave. Questa tabella è introdotta nei programmi durante la fase di compilazione.

Questionario n° 123

Domanda 1: Come passi il tuo tempo libero?

Risposta 1 : Andando al cinema
Risposta 2 : Recandomi al bar con gli amici
Risposta 3 : Leggendo un libro
Risposta 4 : Andando al teatro

Domanda 2: Se vai al cinema con quale criterio scegli il film?

Risposta 1 : Ascolto i consigli degli amici
Risposta 2 : Scelgo quelli del mio regista preferito
Risposta 3 : Leggendo le critiche sul giornale
Risposta 4 : Seguendo i giudizi dei giornalisti

Questionario n° 124

Domanda 1: Quante volte al mese vai fuori Milano?

Risposta 1 : 4 volte
Risposta 2 : 4 volte
Risposta 3 : Nessuna

Domanda 2: Per quale motivo vai fuori Milano?

Risposta 1 : Ho la casa in campagna
Risposta 2 : Ho la casa in montagna
Risposta 3 : Nessun motivo

Domanda 3: Con quale mezzo vai fuori Milano?

Risposta 1 : Andando in macchina
Risposta 2 : Prendo il treno
Risposta 3 : Nessuno

Fig. 5

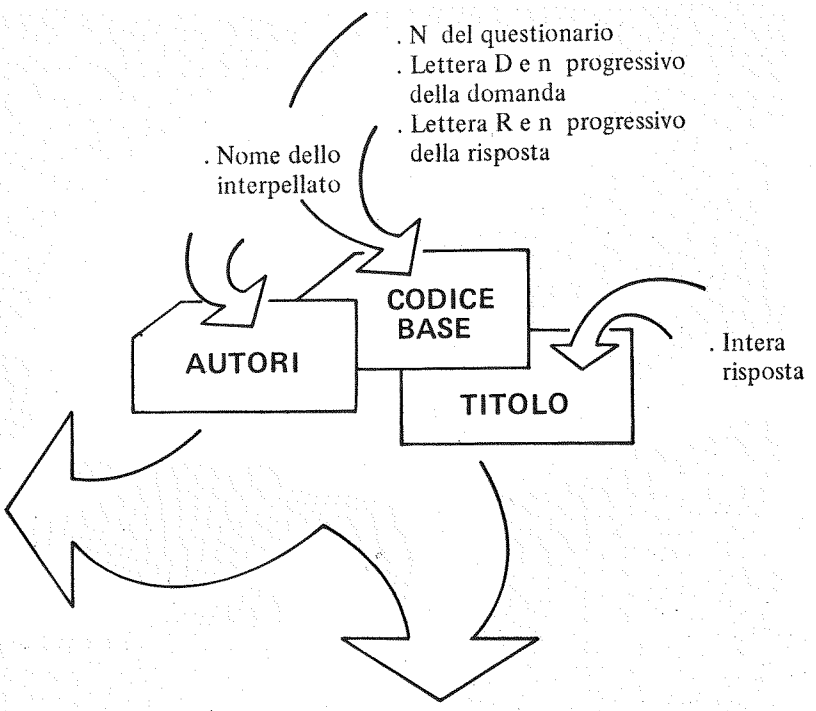
Per rendere l'indice per parole chiave davvero efficiente, cioè per mettere in rilievo solamente le parole più significative, le parole vietate devono essere scelte con cura e secondo precisi criteri, che dipendono sia dall'ambiente al quale è applicato il sistema Kwoc, sia dalla lingua naturale con la quale sono scritti i titoli.

Questo paragrafo descrive appunto un primo studio per determinare la distribuzione delle parole chiave estratte dai titoli dei documenti trattati dal sistema Kwoc. In altre parole cerchiamo di ottenere, mediante l'esame di un insieme di parole chiave estratte dai titoli dei documenti considerati come campione, una legge che rappresenta la relazione fra il numero delle parole chiave che occorrono un determinato numero K di volte e K stesso.

La conoscenza di tale relazione può esserci utile, anche se non nel modo più completo, per valutare se il numero delle parole chiave con una data occorrenza è eccessivo (nel giudizio dei responsabili del sistema) e per considerare eventualmente le parole di tale occorrenza come parole vietate.

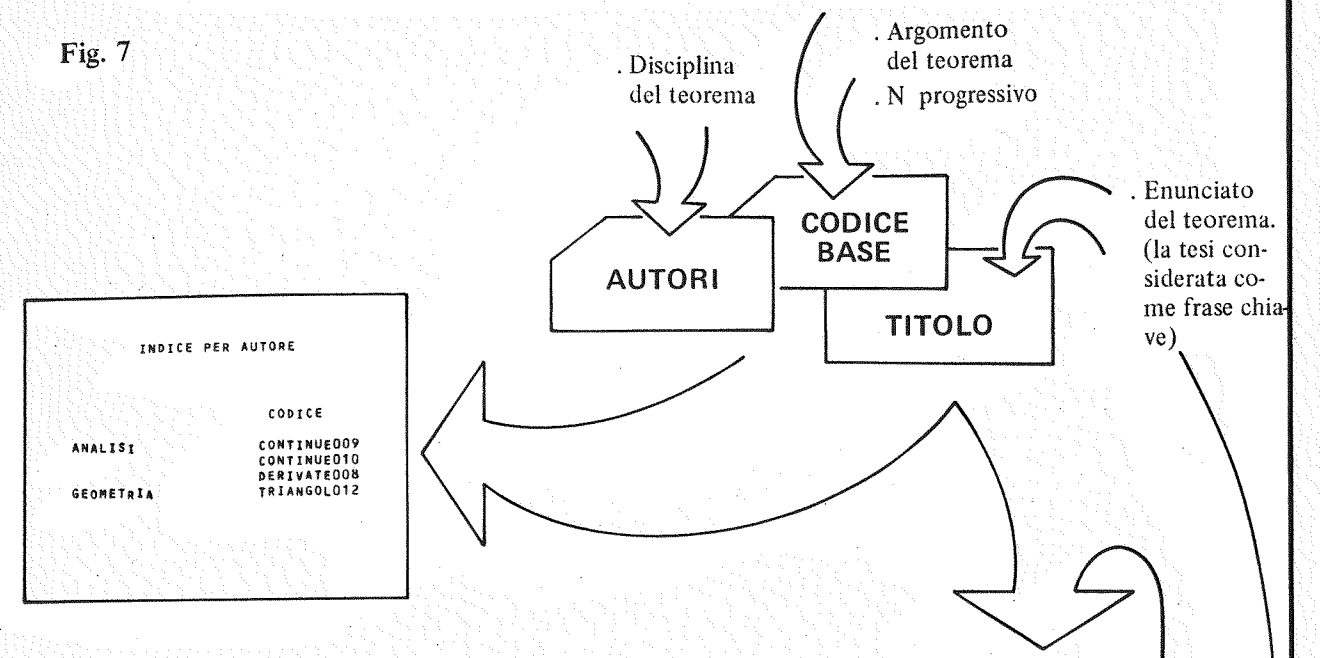
Fig. 6

```
INDICE PER AUTORE

                    CODICE

DOMANDA 1           123D01R0000
DOMANDA 1.          124D01R0000
DOMANDA 2.          123D02R0000
                    124D02R0000
DOMANDA 3.          124D03R0000
PERSONA A.          123D01R0001
                    123D02R0001
                    124D01R0001
                    124D02R0001
                    124D03R0001
PERSONA B.          123D01R0002
                    123D02R0002
                    124D01R0002
                    124D02R0002
                    124D03R0002
PERSONA C.          123D01R0003
                    123D02R0003
                    124D01R0003
                    124D02R0003
                    124D03R0003
PERSONA D.          123D01R0004
                    123D02R0004
QUESTIONARIO 123.   123D01R0000
                    123D02R0000
QUESTIONARIO 124.   124D01R0000
                    124D02R0000
                    124D03R0000
```




```
CODICE                                    INDICE BASE                    UNIVERSITA' DI MILANO    PAGINA   1

123D01R0000   QUESTIONARIO 123.   DOMANDA 1
              1) COME PASSI IL TUO TEMPO LIBERO ?

123D01R0001   PERSONA A.
              ANDANDO AL CINEMA

123D01R0002   PERSONA B.
              RECANDOMI AL BAR CON GLI AMICI

123D01R0003   PERSONA C.
              LEGGENDO UN LIBRO

123D01R0004   PERSONA D.
              ANDANDO AL TEATRO

123D02R0000   QUESTIONARIO 123.   DOMANDA 2.
              2) SE VAI AL CINEMA CON QUALE CRITERIO SCEGLI IL FILM ?

123D02R0001   PERSONA A.
              ASCOLTO I CONSIGLI DEGLI AMICI

123D02R0002   PERSONA B.
              SCELGO QUELLI DEL MIO REGISTA PREFERITO

123D02R0003   PERSONA C.
              LEGGENDO LE CRITICHE SUL GIORNALE

123D02R0004   PERSONA D.
              SEGUENDO I GIUDIZI DEI GIORNALISTI

124D01R0000   QUESTIONARIO 124.   DOMANDA 1.
              1) QUANTE VOLTE AL MESE VAI FUORI MILANO ?

124D01R0001   PERSONA A.
              QUATTRO VOLTE

124D01R0002   PERSONA B.
              QUATTRO VOLTE

124D01R0003   PERSONA C.
              NESSUNA

124D02R0000   QUESTIONARIO 124.   DOMANDA 2.
              2) PER QUALI MOTIVI VAI FUORI MILANO ?

124D02R0001   PERSONA A.
              HO LA CASA IN CAMPAGNA

124D02R0002   PERSONA B.
              HO LA CASA IN MONTAGNA

                                    123D01R0000 * 124D02R0002
```

Fig. 7



INDICE PER AUTORE

	CODICE
ANALISI	CONTINUE009
	CONTINUE010
	DERIVATE008
GEOMETRIA	TRIANGOLO12

CODICE — INDICE BASE — UNIVERSITA' DI MILANO

CONTINUE009	ANALISI OGNI FUNZIONE Y = F(X) DEFINITA IN UN INTERVALLO FINITO I (CHIUSO DI ESTREMI A,B) E IVI MONOTONA, CHE ASSUME ALMENO UNA VOLTA OGNI VALORE COMPRESO FRA F(A) E F(B) E' *CONTINUA NELL'INTERVALLO I$
CONTINUE010	ANALISI SE UNA FUNZIONE E' DERIVABILE IN UN PUNTO X, ESSA E' IVI * CONTINUA $
DERIVATE008	ANALISI SE F(X) E' DERIVABILE NEL PUNTO X1 CON DERIVATA F'(X1), FINITA E DIVERSA DA ZERO, ALLORA ANCHE * LA SUA FUNZIONE INVERSA G(Y) E' DERIVABILE NEL PUNTO Y1 = F(X1) E HA PER DERIVATA L'INVERSO DELLA DERIVATA F'(X1) $
TRIANGOLO12	GEOMETRIA * L'AREA DI UN TRIANGOLO $ E' IL PRODOTTO FRA LA SUA BASE E LA SUA ALTEZZA DIVISO 2

CONTINUE009 * TRIANGOLO12

CODICE — INDICE CHIAVE — UNIVERSITA' DI MILANO — PAGINA 1

TRIANGOLO12	AREA	* L'AREA DI UN TRIANGOLO $ E' IL PRODOTTO FRA LA SUA BASE E LA SUA ALTEZZA DIVISO 2
CONTINUE010	CONTINUA	SE UNA FUNZIONE E' DERIVABILE IN UN PUNTO X, ESSA E' IVI * CONTINUA $
CONTINUE009	INTERVALLO	OGNI FUNZIONE Y = F(X) DEFINITA IN UN INTERVALLO FINITO I (CHIUSO DI ESTREMI A,B) E IVI MONOTONA, CHE ASSUME ALMENO UNA VOLTA OGNI VALORE COMPRESO FRA F(A) E F(B) E' *CONTINUA NELL'INTERVALLO I$
DERIVATE008	DERIVABILE	SE F(X) E' DERIVABILE NEL PUNTO X1 CON DERIVATA F'(X1), FINITA E DIVERSA DA ZERO, ALLORA ANCHE * LA SUA FUNZIONE INVERSA G(Y) E' DERIVABILE NEL PUNTO Y1 = F(X1) E HA PER DERIVATA L'INVERSO DELLA DERIVATA F'(X1) $
DERIVATE008	DERIVATA	SE F(X) E' DERIVABILE NEL PUNTO X1 CON DERIVATA F'(X1), FINITA E DIVERSA DA ZERO, ALLORA ANCHE * LA SUA FUNZIONE INVERSA G(Y) E' DERIVABILE NEL PUNTO Y1 = F(X1) E HA PER DERIVATA L'INVERSO DELLA DERIVATA F'(X1) $
DERIVATE008	FUNZIONE	SE F(X) E' DERIVABILE NEL PUNTO X1 CON DERIVATA F'(X1), FINITA E DIVERSA DA ZERO, ALLORA ANCHE * LA SUA FUNZIONE INVERSA G(Y) E' DERIVABILE NEL PUNTO Y1 = F(X1) E HA PER DERIVATA L'INVERSO DELLA DERIVATA F'(X1) $
CONTINUE009	INTERVALLO	OGNI FUNZIONE Y = F(X) DEFINITA IN UN INTERVALLO FINITO I (CHIUSO DI ESTREMI A,B) E IVI MONOTONA, CHE ASSUME ALMENO UNA VOLTA OGNI VALORE COMPRESO FRA F(A) E F(B) E' *CONTINUA NELL'INTERVALLO I$
DERIVATE008	INVERSA	SE F(X) E' DERIVABILE NEL PUNTO X1 CON DERIVATA F'(X1), FINITA E DIVERSA DA ZERO, ALLORA ANCHE * LA SUA FUNZIONE INVERSA G(Y) E' DERIVABILE NEL PUNTO Y1 = F(X1) E HA PER DERIVATA L'INVERSO DELLA DERIVATA F'(X1) $
DERIVATE008	INVERSO	SE F(X) E' DERIVABILE NEL PUNTO X1 CON DERIVATA F'(X1), FINITA E DIVERSA DA ZERO, ALLORA ANCHE * LA SUA FUNZIONE INVERSA G(Y) E' DERIVABILE NEL PUNTO Y1 = F(X1) E HA PER DERIVATA L'INVERSO DELLA DERIVATA F'(X1) $
DERIVATE008	PUNTO	SE F(X) E' DERIVABILE NEL PUNTO X1 CON DERIVATA F'(X1), FINITA E DIVERSA DA ZERO, ALLORA ANCHE * LA SUA FUNZIONE INVERSA G(Y) E' DERIVABILE NEL PUNTO Y1 = F(X1) E HA PER DERIVATA L'INVERSO DELLA DERIVATA F'(X1) $
TRIANGOLO12	TRIANGOLO	* L'AREA DI UN TRIANGOLO $ E' IL PRODOTTO FRA LA SUA BASE E LA SUA ALTEZZA DIVISO 2

AREA * TRIANGOLO

L'idea base di questo studio è imperniata sulle seguenti considerazioni.

Sia assegnato un multiinsieme (insieme con eventuali ripetizioni) di parole M. Ponendo:

N = numero di parole (con eventuali ripetizioni),
N (K) = numero di parole presenti in M con K ripetizioni,

vogliamo determinare la funzione N (K).

In assenza di informazioni più consistenti, possiamo in un primo momento supporre che la funzione N (K) sia, in qualche modo, in relazione con la legge di Zipf (°), relativa alla frequenza delle parole ordinate.

D'altra parte, niente ci impedisce di formulare l'ipotesi che la funzione sia del tipo esponenziale. Perciò possiamo inquadrare il problema sotto questo punto di vista: verificare se la funzione N (K) si approssima più ad una distribuzione del tipo esponenziale oppure a quella polinomiale (la legge di Zipf), cioè quale delle seguenti ipotesi è più attendibile:

Ipotesi 1: $N(K) = a \cdot e^{mK}$ (1)
Ipotesi 2: $N(K) = t \,/\, K^b$ (2)

Si può dimostrare che l'ipotesi 2, relativamente al caso b = 2, implica il verificarsi della legge di Zipf (°°).

(°) Consideriamo un insieme di parole $V = \{ x_1, x_2, \ldots x_n \}$, e chiamiamo a (x_K) l'occorrenza di x_K (cioè quante volte la parola x_K compare in un dato campione) e l la numerosità del campione, cioè $l = \sum_1^n a(x_K)$.
Se le parole sono ordinate in V secondo la loro frequenza nel campione in ordine decrescente, cioè se vale la condizione $a(x_K) > a(x_i)$ con $K < i$, allora vale la legge di Zipf: $f(K) = \frac{a(x_K)}{l} = \frac{c}{K}$ con c costante, ossia la frequenza della parola di posto K è inversamente proporzionale a K.

(°°) Chiamiamo a (K) il numero di ripetizione della parola di frequenza k-esima.
Ponendo N (K) uguale al numero delle parole che hanno k ripetizioni, dal grafico in figura, che rappresenta l'andamento della legge di Zipf ($a(K) = \frac{\phi}{K}$), possiamo dedurre che:

$$N(K) = a^{-1}(k) - a^{-1}(k+1) \qquad (1)$$

Calcoliamo il 2° membro della relazione (1):

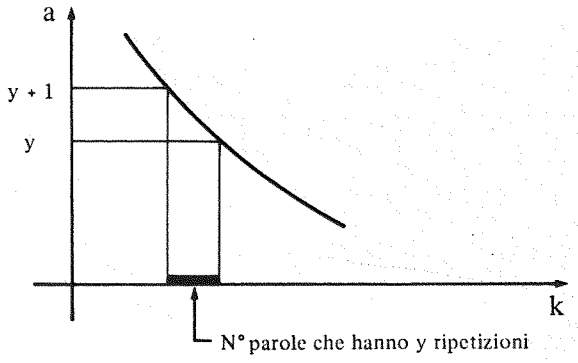
$$a^{-1}(k) - a^{-1}(k+1) = -\frac{a^{-1}(k+\Delta k) - a^{-1}(k)}{\Delta K} \qquad (\Delta k = 1)$$

$$= -\frac{d}{dk} a^{-1}(K) = -\frac{1}{a^1(a^{-1}(K))}$$

Quindi: $$N(K) = -\frac{1}{a^1(a^{-1}(K))} \qquad (2)$$

Utilizzando la legge di Zipf $a(K) = \frac{\phi}{K}$ e sapendo che $a^{-1}(x) = \frac{\phi}{x}$, mediante successive sostituzioni la relazione (2) diventa:

$$N(K) = \frac{\phi}{K^2} \qquad \text{c.d.d}$$



Nella ricerca di soluzione abbiamo seguito questi passi:

1) Siccome le funzioni delle due ipotesi poste non sono lineari, ne consideriamo il logaritmo in modo che le funzioni trasformate così ottenute siano lineari.

2) Calcoliamo i coefficienti di correlazione lineare C_i (i = 1,2) relativamente alle due funzioni trasformate, e di conseguenza la probabilità P_i (i = 1,2) che la relazioni sia lineare.

3) Dal confronto fra le due probabilità valutiamo l'ipotesi più attendibile. Calcoliamo quindi la retta di regressione e la forma della relazione più attendibile.

Sono stati presi come campione 4 tipi di documenti: tesi di laurea italiane, tesi estere, libri, microfilms relativi alle tesi americane.

Il numero dei documenti e delle parole chiave è:

	Numero documenti	Numero parole chiave
Tesi italiane	350	2694
Tesi estere	560	3140
Microfilms	900	5627
Libri	1578	5332

(Ricordiamo che una stessa parola chiave viene contata tante volte quante sono le sue occorrenze nei titoli).

È stato realizzato un programma che, per ogni tipo di documenti, suddivide le parole chiave estratte in vari gruppi a seconda della quantità delle loro occorrenze. È da tener presente che ogni gruppo non si riferisce ad un numero di occorrenze ma ad un intervallo di occorrenze. Così, le parole che compaiono due volte appartengono allo stesso gruppo individuato dall'intervallo con estremi 1 e 2.
Il numero delle parole chiave appartenenti ad ogni gruppo è riportata nella Tabella 1, ove le testate contengono appunto gli estremi degli intervalli.

Sviluppiamo ora i passi sopra elencati, verificandoli con i dati della Tabella 1.

Considerando la relazione $y = a.e^{mx}$ dell'ipotesi 1, passiamo alle coordinate semilogaritmiche, ponendo $Y = \log y$, $X = x$, e abbiamo:

$$Y = \log a + \log e^{mX}$$

cioè: $Y = mX + q \quad \text{con } q = \log a.$

Parallelamente, la relazione $y = \frac{t}{x^b}$ tradotta in coordinate bilogaritmiche diventa:

$$Y = -bX + \log t$$

con $X = \log x,\ Y = \log y.$

Posto $m = -b$ e $q = \log t$ abbiamo:
$Y = mX + q.$

Il secondo passo consiste nel rappresentare graficamente, mediante i dati ottenuti sperimentalmente, le due relazioni trasformate.

Dato che la trasformazione sopra descritta, per quanto riguarda la relazione $y = a.e^{mx}$ (ipotesi 1), ha interessato solamente la variabile y (è stato posto $Y = \log y$), il grafico sarà riportato su scala semilogaritmica. La relazione $y = \frac{t}{x^b}$ sarà rappresentata su scala bilogaritmica.
Le coordinate di ogni grafico si determinano in questo modo:

1) Ascisse: si considera il valore medio dell'intervallo di ogni gruppo della tabella 1.
2) Ordinate: si considera, per ogni gruppo, il rapporto fra il numero delle parole chiave ad esso appartenenti e una variabile L. L è il numero degli interi compresi fra gli estremi dell'intervallo del gruppo, ossia non è altro che il numero delle possibili occorrenze contenute nel gruppo considerato.

In figura 8 e 9 si riportano a titolo d'esempio, 2 grafici relativi ai dati dei microfilms contenuti in tabella 1. Le coordinate di tali grafici sono state determinate adottando il metodo sopra descritto.

Documenti	1-2	3-5	6-10	11-20	21-30	31-40	41-50	51-70
Tesi italiane	912	127	48	13	5	3	0	0
Tesi estere	1104	144	48	20	3	4	0	0
Microfilms	1330	235	87	49	22	5	3	6
Libri	1569	246	90	46	7	7	3	0

Tabella 1

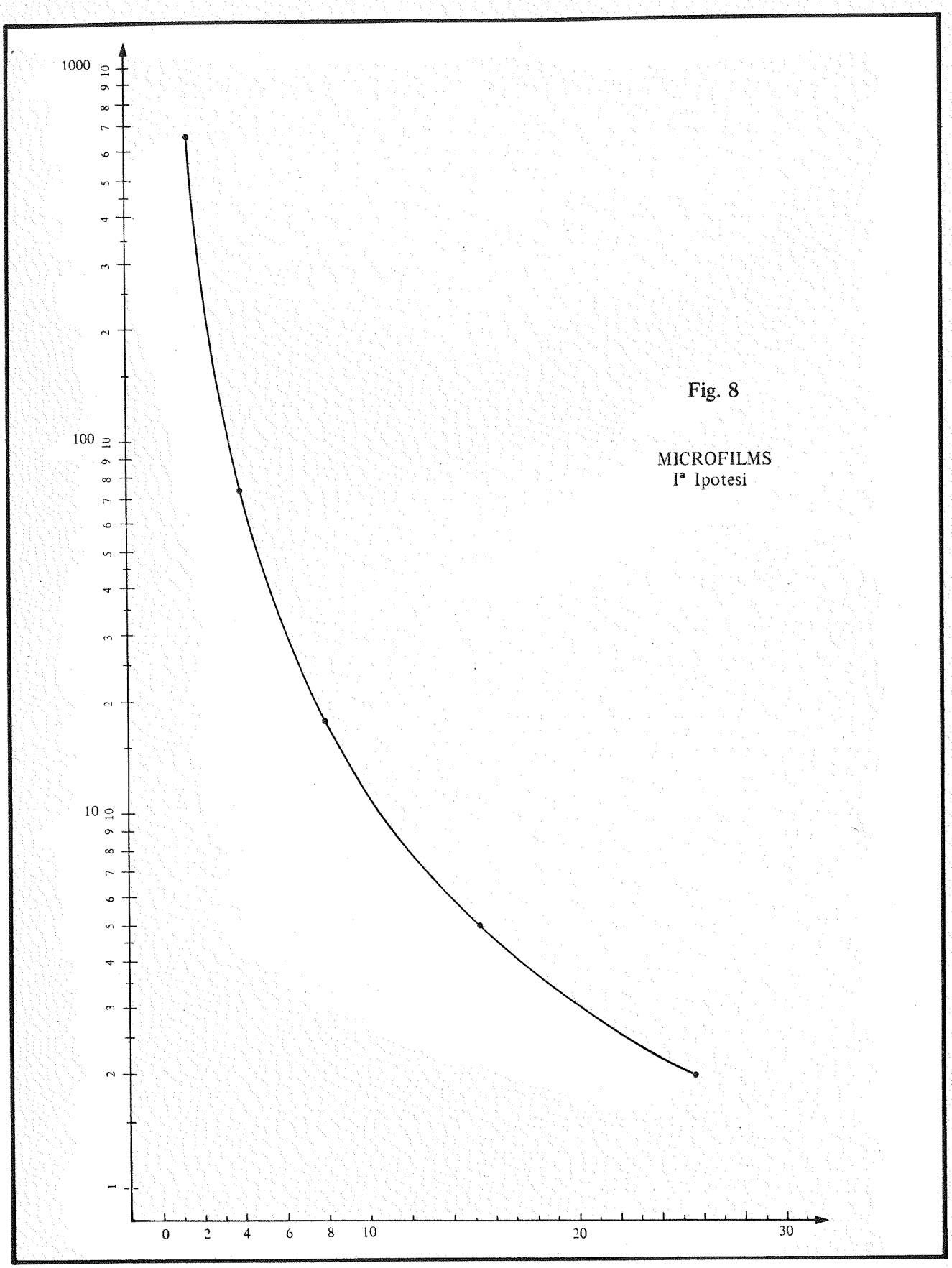


Fig. 8

MICROFILMS
I^a Ipotesi

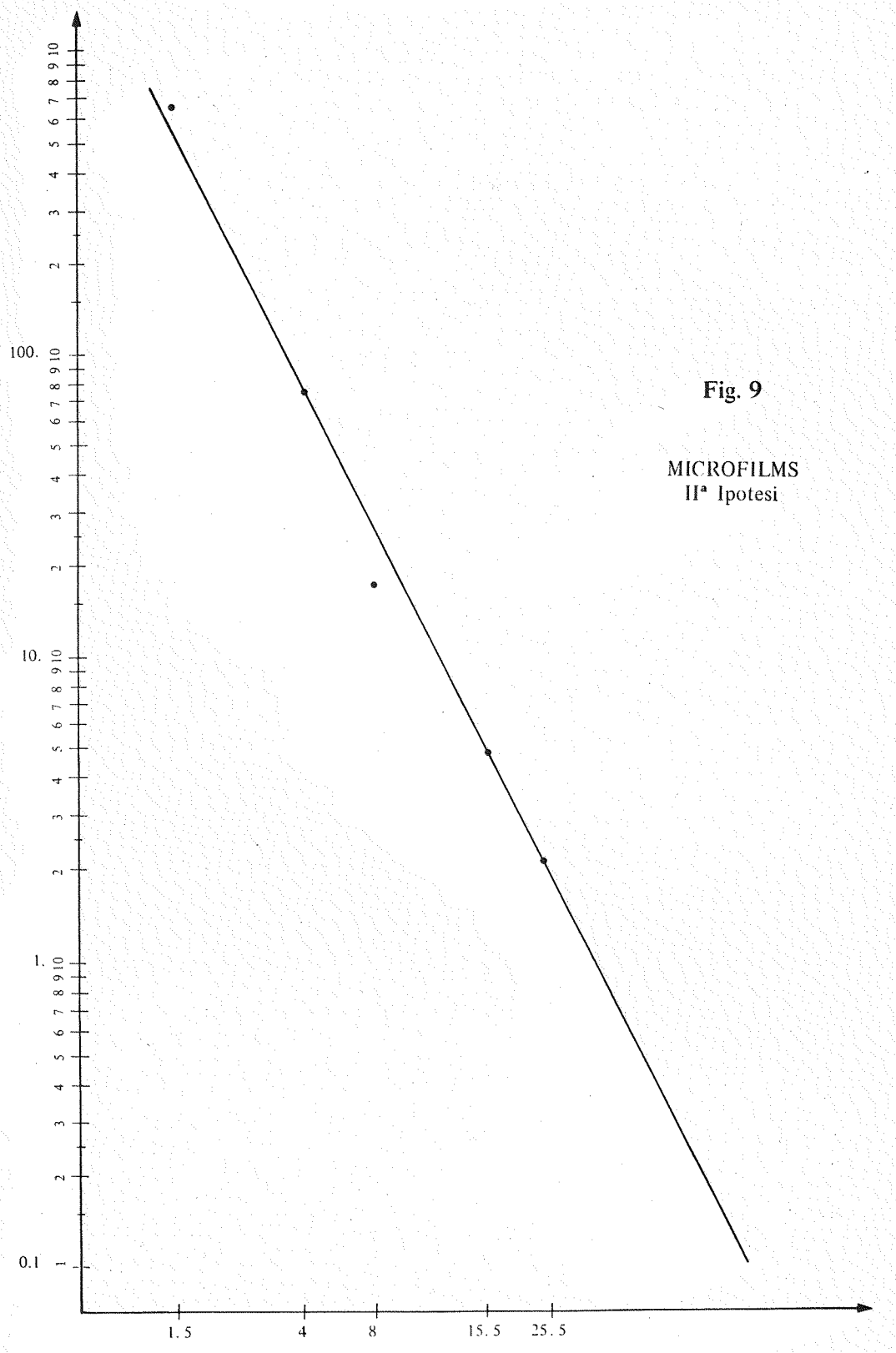


Fig. 9

MICROFILMS
II^a Ipotesi

Il primo grafico è rappresentato su scala semilogaritmica e si riferisce all'ipotesi 1, mentre la scala del secondo grafico, riferito all'ipotesi 2, è bilogaritmica.

Rappresentato un grafico, si misurano, per ogni suo punto, le distanze (con unità di misure arbitrarie, per es. i centimetri) che tale punto ha rispetto all'asse x e all'asse y. Si ottengono così n coppie di distanze (X_i, Y_i), riportate in tabella 2, che serviranno per le successive considerazioni.

di tutto dedurre che l'ipotesi 1 è quella meno attendibile, cioè la distribuzione delle parole chiave non è del tipo esponenziale. D'altra parte i valori dei coefficienti di correlazione della seconda ipotesi giustificano nella deduzione che la distribuzione è del tipo polinominale. Inoltre, considerato il fatto che la relazione $N(K) = {}^{t} / {}_{K}b$ tende alla legge di Zipf per $b = 2$ e valutando i valori di m (con $m = -b$) della tabella, possiamo dire che esiste una certa approssimazione della distribuzione delle parole chiave alla legge di Zipf.

Ascisse	1ª IPOTESI					2ª IPOTESI				
Documenti	0.75	2	4	7.7	12.8	1.1	3.7	5.6	7.5	8.8
Tesi italiane	18.1	13.7	10	5.6	3.9	22.7	16.6	12.6	7.1	4.5
Tesi estere	18.74	13.5	9.9	6.5	2.4	23.5	17	12.5	8.2	3
Microfilms	19	14.4	11.1	8.5	6.6	24	18.1	14.1	10.6	8.5
Libri	19.5	14.6	11.3	8.3	2.4	24.5	18.4	14.2	10.4	5.3

Tabella 2

Cerchiamo ora di stimare, per le due ipotesi, la probabilità che fra le variabili X e Y ci sia una relazione lineare. Determiniamo allora, mediante i dati della Tabella 2 i coefficienti di correlazione con la formula:

$$C = \frac{\sum_{i=1}^{n} X_i Y_i - n\bar{X}\bar{Y}}{\left(\sum_{k=1}^{n} X_k^2 - n\bar{X}^2\right)\left(\sum_{k=1}^{n} Y_k^2 - n\bar{Y}^2\right)}$$

dove $\bar{X} = \frac{\sum X_k}{n}$, $\bar{Y} = \frac{\sum Y_k}{n}$, $k = 1,n$

Si può dimostrare che $-1 < C < +1$ e che per $C = \pm 1$ punti si trovano esattamente su una retta.

Determinati i coefficienti di correlazione, possiamo calcolare la retta di correlazione risolvendo il sistema:

$$\begin{cases} \sum Y_k = m \sum X_k + nq \\ \sum X_k Y_k = m \sum X_k^2 + q \sum X_k \quad (k = 1,n) \end{cases}$$

ottenuto mediante il metodo dei minimi quadrati.

Nella Tabella 3 sono presenti i coefficienti di correlazione C e i coefficienti angolari m della retta di regressione.
Esaminando i dati della Tabella 3 possiamo prima

Documenti	1ª Ipotesi		2ª Ipotesi	
	—C	—m	—C	—m
Tesi italiane	0.86	1.12	0.998	2.38
Tesi ester	0.65	0.96	0.99	2.57
Microfilms	0.92	0.93	0.997	2.016
Libri	0.948	1.295	0.989	2.393

Tabella 3

BIBLIOGRAFIA

[1] B.B Mandelbrot, FRACTALS - FORM, CHANGE AND DIMENSION, Edizioni Freeman G. - S. Francisco, 1977

[2] Mordecai Ezekiel, Kare A. Fox, METHODS OF CORRELATION AND REGRESSION ANALYSIS, Edizioni John Wiley & Sons Inc., 1959

[3] Jordan J.R., KWOC-INDEX - Cobol Programs to Index Bibliographic Information IS, Edizioni Università di Iowa (USA), 2134 Department of Commerce, 1970

L'autore ringrazia vivamente B. Zonta, M. Gasser, A. Bertoni, M. Torelli, e in modo particolare G. Degli Antoni, che insieme hanno dato un valido contributo alla realizzazione di questo lavoro.

SUR LA PROGRAMMATION RATIONNELLE DES ALGORITHMES NUMERIQUES°

A. Bossavit*, B. Meyer**

1. INTRODUCTION

Aux premiers temps de l'expansion du téléphone aux Etats-Unis, la compagnie ATT s'avisa qu'elle ne pouvait continuer dans cette voie fructueuse sans transformer toutes les Américaines en standardistes; et il fallut bien inventer la commutation automatique.

On peut redouter le développement d'une situation analogue parmi les spécialistes du calcul scientifique, dont beaucoup tendent - bien malgré eux - à devenir des programmeurs à plein temps, experts, mais amers.

Or le calcul scientifique n'est ni le seul ni le premier domaine d'application de l'informatique où soient apparus des problèmes de **maîtrise de la complexité** qui, mal surmontés, conduisent à une situation où les difficultés propres à la programmation prennent le pas sur l'analyse des problèmes qu'on cherchait à résoudre en premier lieu.

L'une des causes de cette situation est la coupure trop brutale qui intervient entre les différents niveaux de la résolution d'un problème numérique sur ordinateur. Cette coupure apparaît bien dans la présentation traditionnelle des algorithmes numériques (voir par exemple [3] ou [4]), présentation qui comprend en général deux parties. La première est un exposé mathématique développant l'"idée" qui préside à la méthode employée; elle a les caractéristiques générales de rigueur et de clarté que l'on attend d'une discussion mathématique. La seconde partie - présente seulement dans les articles et les ouvrages qui veulent présenter un aspect "pratique" - fournit un mode d'implantation de la méthode sur ordinateur; elle consiste généralement en un programme, écrit le plus souvent en FORTRAN ou ALGOL, et accompagné de peu d'explications.

L'observation comparée de ces deux étapes ne peut que faire ressortir le caractère intuitif et peu scientifique de la seconde d'entre elles dans les présentations traditionnelles. Un programme FORTRAN cité à l'appui d'une méthode est souvent long, difficile à comprendre de prime abord, truffé de particularités liées par exemple à l'ordinateur sur lequel il a été testé par son auteur; il contient de nombreux choix de représentation implicites (concernant par exemple le mode de rangement en mémoire des éléments d'une matrice, l'affectation d'un même tableau à deux vecteurs qui n'ont pas à être utilisés en même temps, etc.); il s'appuie sur toute une série d'a priori censés être "évidents", et par conséquent non documentés (test de nullité des pivots, produits scalaires en précision étendue, choix d'un traitement par lignes plutôt que par colonnes ou inversement, etc.). Toutes ces décisions de mise en œuvre, cachées et plus ou moins rationnelles, font qu'un bonne dose de foi est nécessaire au lecteur s'il veut se convaincre qu'un programme concret est bien la représentation (l'"implantation") d'une méthode mathématique exposée par ailleurs.

La thèse du présent article est qu'on peut combler, au moins partiellement, ce fossé grâce aux progrès considérables effectués ces dix dernières années en matière de programmation. Nous essayerons de montrer, en nous appuyant sur un exemple de problème numérique, comment on peut obtenir des programmes par une suite de transformations systématiques, et être à même d'accorder au produit final, à sa validité et à sa "solidité", une confiance non pas absolue - là comme ailleurs, l'erreur est humaine - mais beaucoup plus élevée que dans les méthodes traditionnelles.

Nous tenterons de montrer, sur l'exemple bien connu de la factorisation de Choleski, l'intérêt de l'axiomatique des **structures de contrôle**, des méthode de **démonstration de validité** des programmes, et de **l'approche descendante** en programmation des algorithmes numériques. Il conviendrait de compléter ces techniques par l'utilisation des **types abstraits**, non abordés ici.

Les notations employées sont tirées de la référence [2]; les principales sont définies au paragraphe 2. Les autres ne devraient pas poser de difficulté à tout lecteur connaissant un langage de programmation.

* Chef de Division au Département Traitement de l'Information et Etudes Mathématiques, EDF

** Ingénieur chercheur au Département Méthodes et Moyens de l'Informatique, EDF

° Lavoro già pubblicato su "EDF - Bulletin de la Direction des Etudes et Recherches - Serie C - Mathematiques, Informatique, n. 1, 1979, pp. 61-74.

2. NOTATIONS - PRINCIPES DE DEMONSTRATION DE VALIDITE

Nous utiliserons une notation algorithmique [2] destinée à permettre une expression claire des méthodes de résolution des problèmes, et facilement traduisible dans l'un des langages de programmation usuels.

2.1 Programmes et instructions

Un "programme", ou "sous-programme", a la forme générale suivante:

```
programme p (données x, y : ENTIERS,
                    z : REEL;
             résultats h, l : REELS;
             données modifiées m, n, p : ENTIERS,
                    r : LOGIQUE)

| . . . . . corps de programme . . . . .
```

Parmi les arguments, on distingue les "données" (qui ne peuvent être modifiées par *p*), les "résultats" (dont la valeur est calculée par *p*), et les "données modifiées" (dont la valeur initiale est transmise à *p* qui peut la modifier).

Le "corps de programme" est une suite de déclarations et d'instructions séparées par des point virgules.

Parmi les instructions, nous aurons:

1) des affectations, de la forme

variable ← *expression*

2) des instruction conditionnelles, de la forme

```
si condition alors
        | instruction 1
sinon
        | instruction 2
```

ou simplement

```
si condition alors
        | instruction 1
```

où *instruction 1* et *instruction 2* sont des instructions quelconques.

3) des "*blocs*", de la forme

```
| instruction 1 ; instruction 2; . . . ;
| instruction n
```

où *instruction 1, . . . , instruction n* sont des instructions quelconques. L'exécution d'un tel bloc est l'exécution des instructions qui le composent, dans l'ordre.

4) des boucles, de la forme

```
tant que condition répéter
    | instruction
```

dont l'effet est nul si *condition* est vraie, et, sinon, consiste à exécuter une fois *instruction* et à recommencer. Une seconde forme de boucle est:

```
pour i variant de m à n répéter
    | instruction
```

équivalant à

```
| i ← m;
| tant que i ⩽ n répéter
|     | instruction;
|     | i ← i + 1
```

Nous emploierons aussi la notation

```
pour i variant de m à n tant que condition répéter
    | instruction 1
```

équivalant à

```
| i ← m;
| tant que i ⩽ n et condition répéter
|     | instruction 1;
|     | i ← i + 1
```

5) des appels de sous-programmes, de la forme

$p\ (a_1, a_2, a_3, \ldots)$

où a_1, a_2, a_3, . . . sont des objets du programme appelant, de même type que les arguments correspondant définis dans le sous-programme *p*.

2.2 Eléments d'axiomatique des programmes

La présentation précédente est succincte et incomplète. Plus que les notations elles-mêmes, cependant, deux idées sont importantes ici:

1) Le fait que les "structures" définies permet-

tent, par combinaison répétée, de créer des programmes aussi complexes qu'on le désire; ainsi, dans

tant que c répéter
| *a*

a pourra être un bloc, de la forme

| a_1 ; a_2 ;
| a_3

et a_1, à son tour, peut être une instruction conditionelle:

si c alors
| b_1
sinon
| b_2

etc. Le programme résultant, aussi long soit-il, possède une structure simple obtenue par répétion d'un petit nombre de mécanismes de combinaison fondamentaux, et commodément décrite par un arbre.

2) Les instructions de base choisies ont la caractéristique importante de pouvoir être caractérisées par des *axiomes*, permettant ensuite de *démontrer* des propriétés sur les programmes qu'elles servent à construire. Cette idée est l'une des plus importantes qui aient été mises en lumière par les progrès de la méthodologie de la programmation: elle conduit à considérer le texte d'un programme comme un *objet formel*, manipulable dans un certain système mathématique.

L'axiome associé à une instruction I s'écrit sous la forme:

$$\{ P \} \; I \; \{ Q \}$$

où P et Q, écrits sous la forme de commentaires (pour lesquels notre notation utilise des accolades " { " et " } "), sont des assertion caractérisant -en général sous forme de prédicats du premier ordre - des propriétés vérifiées par les objets du programme. Un tel axiome signifie que si l'assertion P, dite *précondition*, est vérifiée, et que I est exécutée, alors Q, dite *postcondition*, sera vérifiée après exécution.

A titre d'exemple, pour toute assertion P, l'affectation $v \leftarrow e$ vérifie

$$\{ P [e \rightarrow v] \} \quad v \leftarrow e \{ P \}$$

où $P [e \rightarrow v]$ désigne la propriété obtenue par substitution de e pour chaque occurence de v dans P.

L'axiomatique complète des structures de contrôle a été développée par Hoare [1]. Nous donnerons ici l'une des règles les plus importantes, celle qui caractérise la boucle ***tant que*** vue plus haut. Cette règle comporte en fait deux parties:

*a) **tant que** c **répéter** A { **non** c }*

En d'autres termes, à la sortie d'une boucle *tant que*, la condition de boucle est fausse (il n'y a pas ici de précondition).

b) $\{ P \text{ et } c \} \; A \; \{ P \} \Longrightarrow$
$\{ P \}$ ***tant que** c **répéter** A* $\{ P \}$

Cette dernière règle indique que si une propriété quelconque P est *invariante* pour une exécution de l'instruction A - du moins dans le cas où la condition de boucle c est vérifiée - alors elle est invariante pour un nombre quelconque d'exécutions de A, donc pour la boucle ***tant que** c **répéter** A*.

La notion d'*invariant de boucle* joue un rôle important dans les démonstrations de validité. A titre d'exemple simple - nous en verrons de plus compliqués au paragraphe suivant -, soit le programme contenant les déclarations

***variables** i : ENTIER,*
somme : REEL;
***tableau** t [1 : 100] : REEL*

et la boucle

| $i \leftarrow 1$; *somme* $\leftarrow 0$;
| ***tant que*** $i \leqslant 100$ ***répéter***
| | *somme* $\leftarrow$ *somme* $+ t[i]$;
| | $i \leftarrow i + 1$

On vérifiera aisément que la propriété P suivante:

$$(i \leqslant 101) \text{ et } \left(\text{somme} = \sum_{j=1}^{i-1} t[j] \right)$$

est bien invariante, dans l'environment $i \leqslant 100$, pour le corps de boucle

| *somme* $\leftarrow$ *somme* $+ t[i]$;
| $i \leftarrow i + 1$

Comme par ailleurs, les actions d'initialisation

$$i \leftarrow 1 \; ; \; \text{somme} \leftarrow 0$$

assurent bien la validité initiale de cet invariant de boucle P, on peut affirmer qu'il reste vrai à la sortie de la boucle. Par ailleurs, la propriété *a)*

nous assure qu'à cette sortie de boucle la condition de boucle $i \leqslant 100$ est fausse, donc $i > 100$. On a donc à la sortie de boucle

$$a)\ i > 100$$

et $$b)\ i \leqslant 101 \text{ et somme } = \sum_{j=1}^{i-1} t[j]$$

c'est-à-dire:

$$i = 101 \text{ et somme } = \sum_{j=1}^{100} t[j]$$

ce qui montre que le programme calcule bien la somme des 100 éléments de t (on aurait pu exprimer le même calcul par une boucle ***pour***, à laquelle se généralise facilement la notion d'invariant).

La présentation des propriétés de la boucle ***tant que*** doit s'accompagner d'une restriction: elles ne sont valables que si la boucle se termine. La finitude d'une boucle ***tant que*** est assurée par l'existence d'un *variant*, soit v, fonction des variables du programme, et tel que

a) $v \geqslant 0$ à l'entrée de la boucle;
b) $\{ v \geqslant 0 \}$ est un invariant de la boucle;
c) chaque exécution du corps de boucle A fait décroître v strictement.

Il est important de noter que les méthodes de démonstration de programmes esquissées ici ont pour intérêt essentiel, non pas de permettre la démonstration *a posteriori* de programmes déjà écrits - exercice d'une utilité limitée -, mais d'influer sur l'écriture des programmes et de favoriser ainsi la fiabilité des produits obtenus. Il s'agit de donner un rôle de tout premier plan aux *assertions* successives, qui sont la justification même des éléments de programme venant de glisser entre elles. Ces assertions visent à expliciter à chaque instant du déroulement d'un programme les propriétés qui doient être vérifiées par les objets de ce programme. Idéalement, la suite des assertions, construite "à l'envers" (en partant de la conclusion) serait écrite avant tout élément de programme; en pratique, on développera concurremment le programme et les assertions, qui forment la base de sa démonstration.

Cette dèmarche exige plus de rigueur, et un développment plus méthodique, que les approches traditionelles. D'une part, en effet, elle oblige à expliciter, sous forme d'assertions mises noir sur blanc, nombre de connaissances et d'hypothèses sur le problème, qui ètaient sans nul doute présentes à l'esprit du programmeur, mais sous forme implicite, dans l'approche usuelle. Il est clair ainsi que l'écriture

$$Y = 1 / (1 - X)$$

suppose l'assertion $\{ X \neq 1 \}$ (ou peut être $\{ |X - 1| > \varepsilon \}$), qu'on aura tout intérêt à exprimer clairement.

L'approche proposée conduit par ailleurs à mettre l'accent sur la notion de *spécification*, en insistant sur la nécessité de définir expressément et aussi complètement que possible les problèmes à résoudre avant de commencer à programmer. L'hypothèse est évidemment que c'est la première phase qui est véritablement difficile, la seconde étant de nature technique (cf. [2]). On considèrera la phase de spécification *statique* comme une activité de programmation à part entière, et le reste du processus comme une suite de transformations systématiques transformant peu à peu la spécification en un programme exécutable.

La méthode proposée tend à diminuer autant que possible la part de l'à peu près et de l'intuition dans le processus de programmation, pour en faire une suite de choix clairs et conscients.

3. UN EXEMPLE DE DEVELOPPMENT DE PROGRAMME

Nous appliquerons les principes précédents, en leur associant l'idée de programmation descendante, à un exemple bien connu, celui de la résolution d'équations linéaires par factorisation de Choleski, en commençant par le cas des matrices triangulaires auquel cette méthode permet de se ramener.

Un mot de précaution s'impose: nous ne prétendons pas que qui ce soit ait inventé cet algorithme de la façon qui va être décrite; s'agissant d'une méthode classique, ce serait évidemment absurde. De la même manière, nul n'affirmera que les grands théorèmes mathématiques ont été découverts à l'origine grâce à des dèmonstrations progressives, rigoureuses et systématiques semblables à celles qu'on trouve dans les manuels. Notre but est méthodologique et pédagogique: la démarche adoptée devrait permettre de retrouver les algorithmes en question

de façon aussi sûre que possible, de bien les comprendre, de bien les faire comprendre, et peut être de se retrouver mieux armé pour l'étude et la recherche de nouveaux algorithmes.

3.1 En quoi la résolution des systèmes triangulaires est-elle "simple"?

Comparons les deux programmes suivants:

```
programme résol (données a : n-MATRICE, b :
                                  n-VECTEUR ;
                 résultats  x : n-VECTEUR,
                                  singulière :
                                  LOGIQUE)
|  si A est régulière alors
|       |  singulière ← faux;
|       |  affecter à x la solution
|       |  de Ax = b
|  sinon
|       |  singulière ← vrai
```

puis *résoltri* qui ne diffère du précédent que par l'adjonction du commentaire initial:

$$\{\, a\,[\,i,j\,] = 0 \quad si \quad j > i \,\}$$

Chacun conviendra que *résoltri* est plus "simple" que *résol*. En fait, la notion de "complexité" ou "simplicité" sous-jacente n'est peut être pas aussi évidente qu'il y paraîtrait à première vue. Essayons de la ramener à la complexité des assertions à satisfaire et des structures de contrôle d'un programme de résolution.

Dans le cas de *résoltri*, nous voulons que soit vraie la propriété

$$(Q) \qquad 0 \leqslant i \leqslant n \Rightarrow \sum_{j<i} a_{ij}\, x_j = b_i$$

à la sortie du programme. L'idée naturelle est de réaliser "progressivement" (Q). Cela signifie par exemple que nous introduisons une variable k dans le programme, destinée à progresser de 0 à n, la propriété

$$(P)\ (0 \leqslant i \leqslant k \Rightarrow \sum_{j<i} a_{ij} x_j = b_i) \text{ et } (0 \leqslant k \leqslant n)$$

devant rester vraie. Or elle est vraie pour $k = 0$. Donc nous cherchons une action A qui, enchaînée avec l'incrementation de k d'une unité, fera de P un *invariant* pour la boucle suivante:

```
tant que k < n répéter
|  k ← k + 1 ;
|  A
```

Alors *résoltri* sera

```
programme résoltri  ( a, b, x, singulière )
|          k ← 0;
|          { P }
|  tant que k < n répéter
|          |  k ← k + 1 ;
|          |  A
|  { P et non ( k < n ) }
```

L'assertion finale signifie que $k \geqslant n$ et que P a lieu, donc que $k = n$ et que Q est bien vérifiée. Une action A satisfaisante est assez facile à trouver:

$$(A) \qquad x_k \leftarrow \Big(b_k - \sum_{j<k} a_{kj} x_j \Big) \Big/ a_{kk}$$

A un détail près, il est clair que $\{\, P \text{ et } C \,\}\ A\ \{\, P \,\}$, et le programme serait correct n'était ce détail : a_{kk} peut être nul; l'action A ne peut alors être exécutée. On peut la remplacer par l'instruction conditionelle suivante, notée A' :

```
(A')
|  si a_kk = 0 alors
|       |  singulière ← vrai
|  sinon
|       |  A
```

et remplacer la condition C, $k < n$, par C' :

(C') $\quad k < n$ *et non-singulière*

On montre alors (par les propriétés de l'instruction conditionelle, que nous n'avons pas introduites):

$$\{\, P \text{ et } C' \,\}\ A'\ \{\, P \,\}$$

et notre programme s'écrit maintenant

```
programme résoltri (a, b, x, singulière)
|  k ← 0;
|  singulière ← faux ;
|  { P }
|  tant que k < n et non singulière répéter
|          |  k ← k + 1 ;
|          |  si  a_kk = 0 alors
|          |       |  A
|          |  sinon
|          |       |  singulière ← vrai
|  { P et non C' }
```

Le programme est donc correct. Comme nous l'avons annoncé à la section 2, la démonstration de validitè n'intervient pas *après* mais s'intègre au processus de construction du programme.

Il ne reste plus qu'à détailler l'action A s'il y a lieu. En FORTRAN, par exemple, la sommation qui figure dans A n'est pas une opération élémentaire et se fait par une boucle.

3.2 La factorisation de Choleski

Soit M une matrice symétrique définie positive. Nous voulons un programme résolvant $Mx = b$. D'après ce qui précède, si l'on sait trouver une matrice S triangulaire inférieure telle que

$$(P) \qquad SS^t = M$$

le programme consistera en l'enchaînement de trois actions:

Factorisation-Choleski de M ;
Résolution de $Sy = b$;
Résolution de $S^t x = y$

et l'on dispose d'un programme pour les deux dernières, puisque S et S^t sont triangulaires. C'est évidemment pour cela qu'on pose a priori P. Notre espoir qu'une telle factorisation de M existe est fondé puisque P équivaut à:

$$(Q) \qquad 1 \leqslant j \leqslant i \leqslant n \Rightarrow \sum_{k=1}^{j} s_{jk} s_{ik} = m_{ij}$$

soit $n\,(n+1)/2$ équations pour $n\,(n+1)/2$ inconnues. Nous considérons donc que Q est la propriété souhaitée à la sortie du programme.

Une technique souvent fructueuse pour construire un programme consiste à partir de la conclusion -ici Q - et à en chercher une version affaiblie qui servira d'invariant de boucle. Ici l'assertion suivante est un bon candidat:

$$(Q') \qquad \begin{array}{l} 0 \leqslant l \leqslant n \\ \text{et } 1 \leqslant j \leqslant i \leqslant l \Rightarrow \displaystyle\sum_{k=1}^{j} s_{jk} s_{ik} = m_{ij} \end{array}$$

L'intérêt de Q' (a priori) vient de ce que:

a) pour $l = 0$, Q' est vraie trivialement
b) pour $l = n$, Q' est équivalente à notre conclusion Q.

Le problème est donc ramené à celui de trouver une action A telle que Q' soit invariante pour la boucle ci-dessous[1] :

tant que $l < n$ ***répéter***
  $l \leftarrow l + 1$;
  A

Le dessin de la figure 1 aide à trouver l'action A :

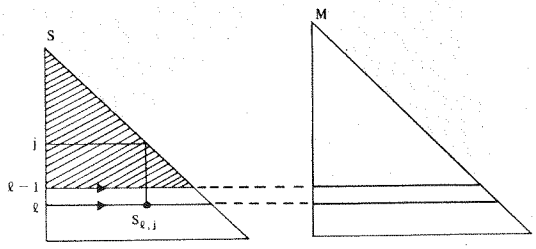


Fig. 1

Lorsqu'on passe de $l - 1$ à l, les égalités suivantes doivent être réalisées *en plus* de celles qui l'étaient déjà au stade $l - 1$:

$$(R) \qquad \left\{ \sum_{k=1}^{j} s_{jk} s_{lk} = m_{lj}, \text{ pour } 1 \leqslant j \leqslant l. \right.$$

On peut raisonner sur R comme sur Q, c'est-à-dire introduire l'invariant de boucle souhaité:

$$(R') \qquad \left\{ \begin{array}{l} 0 \leqslant i \leqslant l \\ \text{et } 1 \leqslant j \leqslant i \Rightarrow \displaystyle\sum_{k=1}^{j} s_{jk} s_{lk} = m_{lj} \end{array} \right.$$

qui redonne R pour $i = l$. On cherche alors l'action B qui, associée à l'incrémentation de i, laisse R' invariant. C'est évidemment:

(1) Toutes les boucles ***tant que*** qui suivent pourraient aussi s'exprimer par des boucles ***pour*** avec progression implicite de l'indice.

$$(B)\quad \left|\begin{array}{l} s_{li} \leftarrow \left(m_{li} - \sum_{k<i} s_{ik}s_{lk}\right)\Big/ s_{ii} \quad \textit{pour } 1 \leqslant i \leqslant l \\ s_{ll} \leftarrow \sqrt{m_{ll} - \sum_{k<l} s_{lk}^2} \quad \textit{pour } i = l \end{array}\right.$$

Ceci nous donne l'action A qui servira à former le corps de la boucle principale:

{ *Action A* }

$i \leftarrow$ { *R' est vrai* } ;
tant que $i < l - 1$ ***répéter*** { *Action B* }
$\left|\begin{array}{l} i \leftarrow i + 1 ; \\ s_{li} \leftarrow \left(m_{li} - \sum_{k<i} s_{ik}s_{lk}\right)\Big/ s_{ii} ; \end{array}\right.$

$s_{ll} \leftarrow \sqrt{m_{ll} - \sum_{k<l} s_{lk}^2}$

{ s_{ll} *n'est pas nul* } { *R est vrai* }

On trouvera page 48 une version FORTRAN de cet algorithme. C'est une traduction littérale, respectant les règles de [2]; la seule modification est l'adjonction d'une variable logique, *DEFPOS*, servant à vérifier au cours des calculs que la matrice initiale est bien définie positive (c'est le principe de la "programmation défensive" et de la "méfiance mutuelle" entre sousprogrammes: lorsque c'est matériellement possible, on évite de croire sur parole une assertion d'entrée).

Le compte d'opérations est le suivant (en remontant des boucles les plus internes vers les plus externes):

On retrouve bien le compte d'opérations traditionnel.

Remarque:

Nous n'avons pas suivi la présentation traditionelle de l'algorithme. Sans doute par imitation de l'elimination de Gauss, on travaille en général "par colonnes": supposant connu le trapèze de la figure 2, on montre qu'il est "facile" de calculer la colonne l.

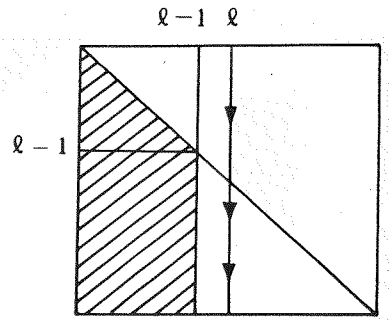


Fig. 2

Pour retrouver cette variante, revenons à notre conclusion Q. Il entrait un certain arbitraire dans le choix de l'invariant de boucle Q'. Il serait aussi légitime de choisir l'invariant

$$(Q'_l)\quad \left|\begin{array}{l} 0 \leqslant l \leqslant n \\ \textit{et } j \leqslant i,\ 1 \leqslant j \leqslant l \quad \sum_{k=1}^{j} s_{jk}s_{ik} = m_{ij} \end{array}\right.$$

	Multiplications	Division	Extractions de racines
Action B Boucle sur l'action B (le calcul précedént pour i variant de 1 à $l-1$)	$i - 1$ $\frac{(l-1)(l-2)}{2}$	1 l	
Action A (la boucle précédente plus le calcul de s_{ll})	$\frac{(l-1)(l-2)}{2} + l - 1 = \frac{l(l-1)}{2}$	l	1
Ensemble du programme (l'action A pour l variant de 1 à n)	$\sum_{l=1}^{n} \frac{l(l-1)}{2} \simeq \frac{n^3}{6}$	$\frac{n(n+1)}{2}$	n

"libérant" en quelque sorte l'indice *i*. On trouve aisément le corps de boucle correspondant:

$$
\begin{array}{l|l}
 & s_{ll} \leftarrow \sqrt{m_{ll} - \sum_{k<l} s_{ik}^2}\,; \\
(A_1) & \textbf{\textit{pour tous les }} i \textbf{\textit{ tels que }} l < i \leqslant n \textbf{\textit{ répéter}} \\
 & \quad s_{il} \leftarrow \left(m_{il} - \sum_{k<l} s_{ik} s_{lk} \right) \Big/ s_{ll}\,;
\end{array}
$$

On écrirait facilement le programme correspondant [4].

Ces variantes doivent être étudiées de près dès qu'on se pose des problèmes de représentation physique (grande matrice rangée par lignes), en particulier en liaison avec les problèmes de pagination en mémoire virtuelle. La question des structures de données amène rapidement à s'intéresser à un autre aspect des recherches récentes en programmation: la théorie des *types abstraits*, qui permet de distinguer nettement les propriétés externes des objets manipulés (des matrices dans le cas qui nous occupe) des problèmes de représentation physique.

4. CONCLUSION

Qu'avons-nous obtenu? Certainement pas un algorithme révolutionnaire. Nous avons "pris du recul" par rapport au problème de la factorisation de Choleski, et essayé de ne pas mettre la charrue avant les bœufs: il est malsain de commencer à coder un programme à l'aveuglette, en suivant son intuition ou l'habitude, sans avoir posé clairement le problème à résoudre. De l'exposé précis de ce problème, nous avons vu qu'il était possible de déduire un invariant de boucle, puis la boucle elle-même et le programme. L'invariant n'est pas déterminé de façon univoque; différents choix, tous justifiables, mènent à différents algorithmes; la démarche adoptée permet de comprendre en profondeur, nous semble-t-il, les points communs de ces algorithmes et leurs différences réelles.

Enfin, la démarche même de construction de l'algorithme assure un degré de confiance en sa validité, certes pas absolu, mais sans aucun doute supérieur à tout ce que l'on peut attendre des méthodes traditionnelles.

Une réflexion méthodologique de cet ordre, et son application à l'enseignement dans l'Université et les Grandes Ecoles, nous paraissent nécessaires si l'on

```
      SUBROUTINE CHOFAC (N, NMAX, M, S, DEFPOS)
      INTEGER N, NMAX
        REAL M (NMAX, N), S (NMAX, N)
C  FACTORISATION DE CHOLESKI DE LA MATRICE M, M = S*TRANSP (S)
C  SI ELLE EST POSSIBLE, DEFPOS AURA LA VALEUR . TRUE., SINON, . FALSE.
C  VARIANTE PAR LIGNES, MATRICES PLEINES.
C  NMAX = DIMENSION DES COLONNES DE M ET S DANS L'APPELANT.
      DEFPOS = . TRUE.
      L = 0
C     /TANT QUE L < N ET DEFPOS REPETER/
1     IF ( (L.GE.N) . OR . (.NOT. DEFPOS) )GOTO 9
        L=L + 1
        I = 0
C       /TANT QUE I < L REPETER/
2       IF (I.GE.L) GOTO 3
          I = I + 1
C         -PRODUIT SCALAIRE DES LIGNES I ET L :-
            SOM = M (L, I)
            K = 0
C           /TANT QUE K < I REPETER/
3           IF (K.EQ.I) GOTO 4
              K = K + 1
              SOM = SOM - S (I, K)*S (L, K)
              GOTO 3
C           /FIN TANT QUE/
4           CONTINUE
C         /SI I = L DIVISER/
          IF (I.EQ.L) GOTO 5
            S (L, I) = SOM / S (I, I)
            GOTO 7
C         /SINON SI SOM < = 0 MATRICE NON DEFINIE POSITIVE/
5         IF (SOM. GT. 0.) GOTO 6
            DEFPOS = . FALSE.
            GOTO 7
C         /SINON TERME DIAGONAL = RACINE CARREE/
6           S (L, L) = SQRT (SOM)
C         /FIN SI/
7         GOTO 2
C       /FIN TANT QUE/
8       GOTO 1
C     /FIN TANT QUE/
9     RETURN
      END
```

veut éviter aux spécialistes du calcul scientifique d'être submergés à court terme par les difficultés matérielles de la programmation, jusqu'à devenir des "opérateurs de la programmation" comme il existe des "opérateurs du téléphone".

BIBLIOGRAPHIE

[1] Hoare C.A.R. - AN AXIOMATIC BASIS FOR COMPUTER PROGRAMMING. Communications of the ACM, 12, 10, pp. 576-583 (octobre 1969).

[2] Meyer Bertrand et Baudoin Claude. - METHODES DE PROGRAMMATION. Paris: Eyrolles 1978.

[3] Wilkinson J.H. - THE ALGEBRAIC EIGENVALUE PROBLEM. Oxford: Clarendon Press 1965.

[4] Wilkinson J.H. et Reinsch C. - LINEAR ALGEBRA (HANDBOOK FOR NUMERICAL COMPUTATION, VOL. II). Berlin: Springer-Verlag 1971.

AVVENIMENTI

NOTE A MARGINE DEL WORKSHOP "STANDARDS FOR SYSTEM ANALYSIS" SVOLTOSI A ROMA NEL MARZO 1980

Quello che segue non è il resoconto del Workshop "Standards for System Analysis" organizzato dallo I.A.G. (IFIP Applied Information Processing Group) il 19, 20 e 21 marzo 1980[1], ma un insieme di considerazioni che mi sono state sollecitate, durante il seminario e dopo, dall'avervi partecipato.

Non ho quindi l'obiettivo di fare un discorso completo ed esauriente ma solo di proporre alcuni temi di riflessione.

• La fase di analisi nel ciclo di vita di un sistema informativo è ormai riconosciuta come di importanza vitale nella produzione dei sistemi informativi e, di conseguenza, del software che li supporterà, ma è ancora quella su cui le conoscenze sono meno sistematizzate e maggiore è la confusione di idee, linguaggi, culture.

Il significato stesso della parola *analisi* è oggetto di controversie (di cui si è sentita l'eco anche al Workshop): c'è chi vuole la fase di analisi ridotta alla sola attività che conduce alla definizione dei requisiti e chiama le fasi successive *disegno architetturale, disegno dettagliato*, ecc. (mutuando così in questo campo la distinzione tra analisi e sintesi corrente in Teoria dei Sistemi[2]) e chi chiama analisi tutto quello che non è programmazione (riproducendo così nei nomi delle attività la tradizionale distinzione tra analisti e programmatori).

Buckle ha proposto nella sua relazione una denominazione intermedia tra le due citate sopra, distinguendo due fasi nell'analisi: *l'analisi dei requisiti* e *l'analisi funzionale*, e riducendo la fase di disegno a quella che sopra abbiamo chiamato di disegno dettagliato (vedi più avanti figura 1).
Questa proposta, se accettata come standard lessicale, sarebbe un utile passo verso la semplificazione della discussione sull'analisi. Ma non è questo il punto che mi preme discutere, quanto piuttosto un problema a monte di esso.

Quali sono le cause di questa diversità di lessico, o meglio, che cosa indica questa diversità di lessico? A me pare che essa sia un indizio, non il solo certamente, di un problema più complesso.

La fase di analisi, per essere quella più a monte del ciclo di vita di un sistema informativo, vede giustapporsi (ma raramente fondersi) diverse culture portate da diversi soggetti: la cultura caratteristica dell'organizzazione il cui sistema informativo deve essere modellato o modificato (e comunque analizzato) che è portata con attitudini diverse dalla direzione generale che commissiona l'intervento e dagli utenti finali il cui lavoro sarà influenzato dall'intervento stesso, la cultura organizzativa portata dal consulente d'organizzazione o dall'ufficio di organizzazione, la cultura specifica dell'ingegneria dei sistemi informativi, portata da chi deve disegnare la struttura del sistema, e la cultura più propriamente informatica portata da chi realizza il software che implimenterà le procedure automatizzate e gli archivi di dati del sistema[3].

La babele di linguaggi e di idee non potrà trovare che soluzioni parziali, settoriali e contingenti finchè non si sarà affrontato il problema di quale può e deve essere il terreno comune su cui queste culture si incontrano.

(1) Gli atti del Workshop sono in preparazione e forniranno una informazione dettagliata delle tematiche discusse. Ci basta qui ricordare che ad esso hanno partecipato una ventina di persone (consulenti, ricercatori universitari, responsabili di standard e metodologie e dirigenti di settore EDP di grossi utenti) provenienti da diversi paesi europei. I lavori si sono sviluppati in una serrata discussione di due relazioni tenute da Giovanni Damaggio della SINCON (un inquadramento generale della problematica dell'analisi ed una riflessione sul ruolo e sul significato degli standard in questo campo) e da John Buckle della Università di Oxford (una definizione dei requisiti che gli standard di analisi devono soddisfare per essere efficaci sulla base di una discussione del ciclo di vita di un sistema informativo).
L'informalità della discussione, le diverse esperienze dei partecipanti hanno fatto sì che il dibattito è stato vivace e stimolante.

(2) Si veda ad esempio Klir [1]

(3) Su questo argomento ho svolto considerazioni più articolate nella Giornata di lavoro su Problemi di Definizione di Requirements, di Analisi e di Disegno di Sistemi Software, svoltasi nell'ambito del Progetto Finalizzato per l'Informatica del CNR P1 METOD [2].

Buckle, nella sua relazione, ha tratteggiato a grandi linee un confronto tra l'ingegneria dei sistemi informativi e altre branche più consolidate della ingegneria (ingegneria dei ponti ed ingegneria aeronautica): credo che sia una strada che, sviluppata, può dare spunti interessanti per la soluzione del problema sopra indicato.

• Damaggio, nella sua relazione, ha affermato che non c'è differenza tra sviluppo e manutenzione di un sistema informativo. È una considerazione significativa che mette in luce come la concezione ex novo di un sistema informativo sia un caso limite, raro ed eccezionale, se si pensa che si interviene in una struttura preesistente in cui un sistema informativo (magari non automatizzato) è già operante[4]. Essa sottolinea che va evitata l'identificazione del sistema informativo con la sua parte automatizzata.

L'affermazione di Damaggio ha trovato una conferma nel disegno del ciclo di vita proposto da Buckle (figura 1), che merita alcune considerazioni.

— Il ciclo di vita è chiuso: ciò mette bene in luce che ogni nuovo intervento è destinato a modificare una situazione non a creare un sistema informativo (il termine manutenzione che appare nel disegno associato ad installazione deve intendersi perciò come *piccola manutenzione* del software e non nel senso che Damaggio gli attribuiva).

— Per ogni fase vengono identificate l'*attività realizzativa* (la freccia che va in senso orario) il *documento* o i documenti realizzati (il nodo del grafo) e l'*attività di controllo* del risultato realizzato in quella fase rispetto alle fasi precedenti (la freccia che va in senso antiorario). Quest'ultima è assai importante perchè impone l'identificazione delle verifiche necessarie in ogni fase per la validazione del risultato raggiunto. Emerge inoltre l'aspetto contrattuale del controllo del documento di definizione dei requisiti del sistema, da cui deriva la centralità della comunicazione con l'utente nella fase di analisi.

— La freccia che connette *sistema all'opera* con *bisogni e idee dell'utente* è l'unica freccia non orientata. Questo per indicare che ciò che è stato indicato con *qualità del prodotto* non è un'attività il cui risultato va in altro modo verificato ma una interazione tra sistema e utente, in cui l'utente verifica se il sistema risponde alle sue aspettative, ma il sistema può, operando, modificare idee e bisogni dell'utente. È questa interazione quindi, che produce una più chiara definizione dei bisogni da parte dell'utente, o anche l'emergere di nuovi bisogni, e rimette in moto il ciclo.

— Questa rappresentazione del ciclo di vita descrive bene come ordinare le attività necessarie per la concezione e la manutenzione di un sistema informativo nella situazione attuale, in cui il ruolo dell'utente è essenzialmente passivo, e comunque mal definito e mal gestito. In una situazione più evoluta in cui l'utente/committente sarà in grado di esprimere e controllare le qualità dell'intervento che richiede, sarà probabilmente necessario distinguere due cicli, entrambi proiezione di quello proposto in figura 1: il ciclo di vita del sistema informativo che riguarda l'utente/committente ed è chiuso ed il ciclo di produzione delle singole modifiche (più o meno ampie e che comportano la creazione di software) che riguarda il gruppo di progetto chiamato a realizzarle, ed è aperto. In un quadro di questo genere sarà possibile, secondo me, che la descrizione del comportamento desiderato del sistema (le *funzioni* in figura 1) diventi strumento di controllo del funzionamento del sistema[5].

• Al Workshop si è parlato poco dei metodi di analisi esistenti e dei linguaggi che questi propongono per descrivere il comportamento dei sistemi. Ciò è dovuto a cause che è interessante cercare di interpretare.

Una prima causa può essere identificata nel fatto che la diffusione delle metodologie di analisi è ancora ridotta, che siamo ancora nella fase in cui emerge con forza un bisogno di strumenti nuovi per l'analisi, ma non si hanno ancora, salvo rare e interessanti eccezioni, esperienze consolidate di utilizzo delle metodologie esistenti.

Una seconda causa può essere invece trovata nel fatto che i linguaggi proposti nelle metodologie esistenti sono in larga misura insoddisfacenti e inadeguati.

(4) Winograd propone una riflessione analoga sulla stessa attività di programmazione: "La principale attività della programmazione non consiste nella concezione di nuovi programmi indipendenti, ma nella integrazione, modifica e spiegazione di quelli esistenti" [3].

(5) Degli Antoni, Maiocchi, Polillo e Zonta [4] hanno recentemente sviluppato considerazioni articolate e suggestive interpretando la descrizione del comportamento desiderato del sistema (i requisiti) come sorgente del controllo sul sistema stesso.

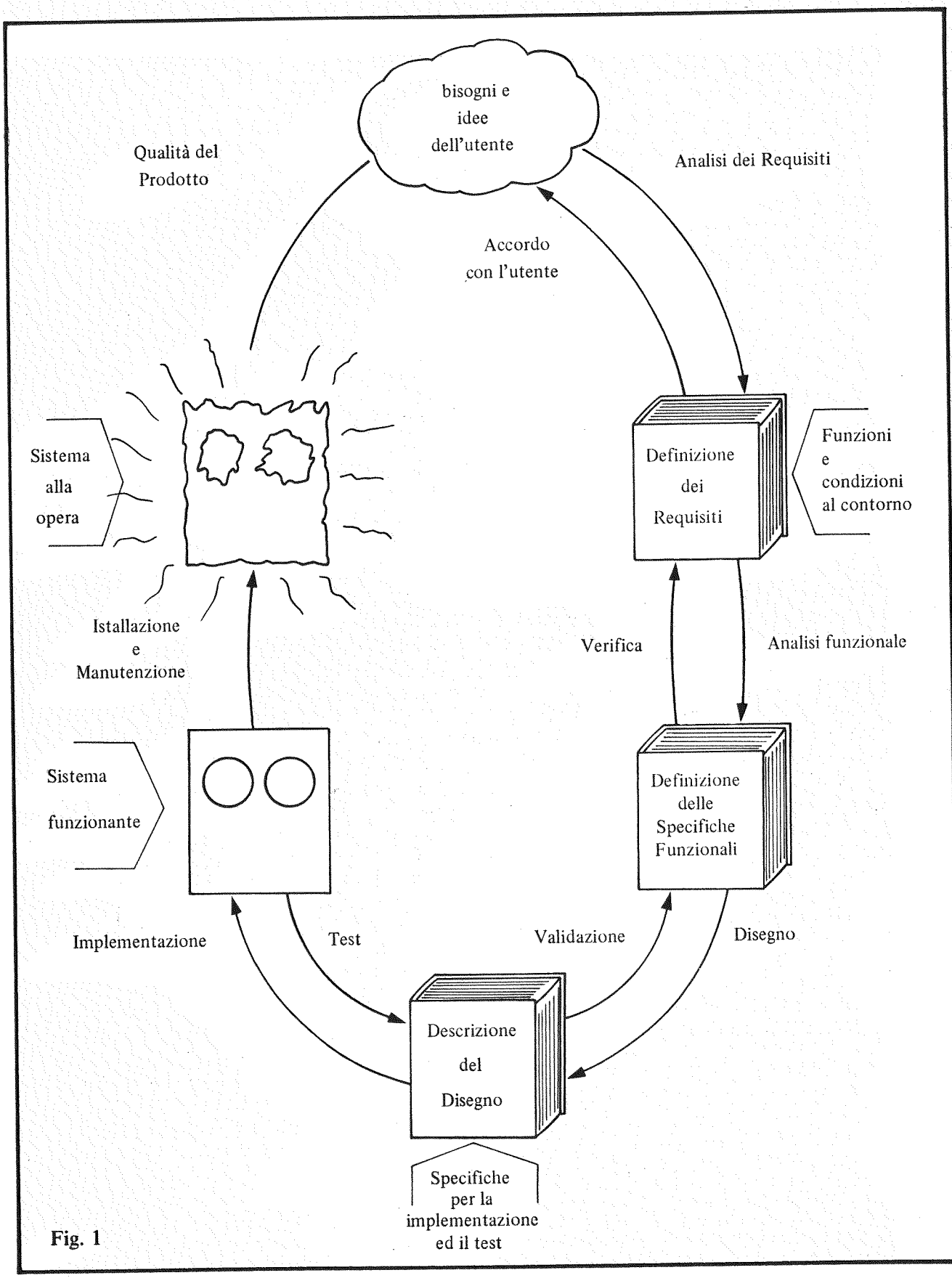

bisogni e
idee
dell'utente
Qualità del
Prodotto
Analisi dei Requisiti
Accordo
con l'utente
Sistema
alla
opera
Definizione
dei
Requisiti
Funzioni
e
condizioni
al contorno
Istallazione
e
Manutenzione
Verifica
Analisi funzionale
Sistema
funzionante
Definizione
delle
Specifiche
Funzionali
Implementazione
Test
Validazione
Disegno
Descrizione
del
Disegno
Specifiche
per la
implementazione
ed il test

Fig. 1

Durante il workshop è emersa, infatti, una contraddizione tra chi affermava che l'uso di linguaggi formali nell'analisi dei requisiti è reso, se non impossibile, assai difficoltoso dal problema dell'interazione con l'utente (che non è in grado non solo di fare uso di un linguaggio formale ma neppure di utilizzare la documentazione con esso prodotta), e chi affermava che l'adozione di un linguaggio formalizzato è comunque necessaria se si vuole che la definizione dei requisiti possa servire per verificare effettivamente se il sistema risponde ai bisogni dell'utente. Questa contraddizione mette in luce un nodo a cui è impossibile dare risposte univoche e immediate. Infatti, al di là della consigliata adozione di linguaggi grafici, è comunque necessario immaginare un processo di formazione degli utenti, che integri la loro cultura con una capacità di interpretare e descrivere il comportamento del sistema in cui sono inseriti.

Ad ostacolare il decollo di un processo di questo genere contribuiscono non poco, a mio avviso, le caratteristiche dei linguaggi che vengono proposti nella grande maggioranza delle metodologie oggi esistenti.

Da una parte, infatti, abbiamo linguaggi che sono varianti di linguaggi di programmazione, orientati alla cultura dell'analista e del programmatore e che non contribuiscono in alcun modo a rompere il diaframma che divide questi dagli utenti. Dall'altra, abbiamo linguaggi assai poveri come potenza espressiva: vengono proposti diagrammi a blocchi, grafici più o meno complicati che non evidenziano i problemi organizzativi che l'intervento in corso dovrebbe risolvere. Questi vengono allora descritti a parte, per mezzo di tabelle, di documenti in linguaggio naturale, ecc., e non viene messo in luce l'intreccio tra descrizione del comportamento del sistema e individuazione dei problemi da risolvere, che sarebbe uno stimolo assai forte all'adozione di un linguaggio formalizzato anche da parte dello utente(6).

La principale carenza dei linguaggi generalmente adottati nelle metodologie di analisi oggi esistenti risiede, a mio avviso, nella loro inadeguatezza a descrivere la concorrenza (sia essa determinata da comunicazione o cooperazione) tra diversi processi: è infatti nell'interazione tra sottosistemi che nascono i più grossi problemi organizzativi.
Per questo motivo ritengo le reti di Petri, e comunque i linguaggi basati su insiemi di coppie < condizione, azione > come i guarded commands e i sistemi di produzione, lo strumento più adatto per fornire descrizioni formalizzate del comportamento di sistemi(7).

• Damaggio, nella sua relazione, ha osservato che gli standard hanno due funzioni: una *normalizzatrice*, per creare una uniformità nei metodi di lavoro e nelle tecniche di documentazione che faciliti il controllo del gruppo di progetto, ed una *metodologica*, per fornire strumenti che facilitino la adozione di "buone scelte" nelle varie fasi del processo di concezione e manutenzione dei sistemi informativi. Questi due obiettivi vanno comunque sempre visti intrecciati: se assunti indipendentemente l'uno dall'altro possono condurre a scelte contradditorie e/o fallimentari.

È necessario allora, da una parte, definire esattamente che cosa si intende con standard (in opposizione a altri termini come guideline, methodology, tool) e, in secondo luogo, indicare una strategia per l'adozione di standard.
Nel dibattito del Workshop si è usata la parola *standard* in opposizione a *guideline*, intendendo con standard una regola (od un insieme di regole) tassativa per chi coopera al progetto, con guideline invece una regola semplicemente consigliata.

La distinzione mi sembra poco chiara. Mi convince di più dire che uno standard è una prescrizione che fissa le azioni da compiere ed il modo in cui queste vanno eseguite, mentre una guideline è una indicazione delle azioni da compiere che lascia margini di libertà nella scelta del modo con cui tali azioni vanno realizzate.

Una metodologia, quindi, sarà composta in genere di standard (ad es.: adozione del linguaggio dei flowchart, derivazione del programma dai dati secondo le prescrizioni del metodo Jackson, ...) e di guideline (ad es.: utilizzo di nomi evocativi, costruzione del programma in modo top-down, ...).

(6) Una rassegna sommaria dei linguaggi di descrizione di processo è reperibile nello studio esplorativo sui metodi di descrizione dei processi amministrativi realizzata nell'ambito del Progetto Finalizzato per l'Informatica P2.A dall'unità operativa dello Istituto di Cibernetica dell'Università di Milano operante sullo obiettivo PROCAM [5].

(7) In [4] si argomenta più diffusamente questa tesi: "Gli autori suggeriscono con forza, per tutte le ragioni indicate, che l'ambito concettuale, in cui il comportamento desiderato dei sistemi attuali deve essere rappresentato, va ricercato in seno alla Teoria Generale delle Reti sviluppate da Petri, Holt e altri".

Se intendiamo in questo senso standard e guideline appare evidente che, rispetto ai due obiettivi attribuiti da Damaggio all'adozione di standard, quelli di natura metodologica trovano risposta in guideline, mentre quelli di carattere normalizzatore la trovano in standard.
Sarebbe però un errore pensare in questi termini separati: ciascun problema metodologico e di controllo non deve trovare una risposta ad hoc, indipendente dagli altri standard e guideline adottati, ma impone una riconsiderazione del *sistema* di standard e guideline che costituisce la metodologia.

Solo in questo modo si evita la proliferazione di prescrizioni che, invece di migliorare la produttività, burocratizzano il processo lavorativo.

Si è detto al workshop, a questo proposito, che, se è pur vero che gli standard aumentano la produttività, non è per questo vero che vale l'equazione: più standard = più produttività.

Giorgio De Michelis

Riferimenti

[1] G.J. Klir, AN APPROACH TO GENERAL SYSTEMS THEORY, Van Nostrand Reinhold Company, New York 1969

[2] Atti Giornata di Lavoro su: PROBLEMI DI DEFINIZIONE DI REQUIREMENTS, DI ANALISI E DI DISEGNO DI SISTEMI SOFTWARE, Istituto di Cibernetica, Univ. di Milano, METOD 15, 1979

[3] T. Winograd, BEYOND PROGRAMMING LANGUAGES, Comm. ACM, 22.7, luglio 1979

[4] G. Degli Antoni, M. Maiocchi, R. Polillo, B. Zonta, HOW, WHAT WHY - Proc. IV Annual Honeywell International Software Conference, Minneapolis, 1980

[5] Studio esplorativo su: METODI DI DESCRIZIONE DEI PROCESSI AMMINISTRATIVI, Istituto di Cibernetica, Univ. di Milano, Prog. Fin. per l'Informatica, P2.A, Obiettivo PROCAM, 1980

IL SEGNALIBRO

In questa rubrica vengono segnalati libri, articoli o studi di recente pubblicazione che, a giudizio della redazione o dei lettori di NOTE DI SOFTWARE, rivestono un particolare interesse nell'ambito dei temi a cui questa rivista è dedicata.
Le citazioni fra virgolette, se non ne è indicata la fonte, sono tratte dai lavori stessi.

Ingegneria del software - un libro

• Jensen, R.W., Tonies, C.C., SOFTWARE ENGINEERING, Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, 1979, pp.xi + 580

"[...] This textbook represents the culmination of many studies delving into the nature and methods of software development. This text presents software engineering, not as an isolated software design methodology or as a simple set of development tools and techniques, but as a comprehensive, broad, problem-solving discipline very similar to other branches of engineering. Our coverage includes tha management, legal, and security aspects as well as the technical aspects of the subject (structural design, structured programming, and verification and validation techniques), since we feel it is also necessary for software engineers to be conversant in all aspects of software engineering to be effective in their tasks. We suggest that you examine the proposed curricula topics listed in the Appendix to obtain a picture of the educational requirements for software engineers. This book is intended to be used as a textbook either in a comprehensive software engineering course or in a sequence of courses covering this broad range of software engineering topics, or as a single, software engineering reference source. Each of the chapters in this text is intended to serve as an introduction to the topic covered by that chapter, as well as a reference source for those already familiar with the subject material. Each chapter is essentially self-contained and can be presented as an individual unit, independent of the other chapters.[...]"

Indice: 1. INTRODUCTION, by R.W. Jensen, C.C. Tonies: 1.1 The Software Crisis, 1.2 The Industrial Environment, 1.3 Software Engineering, 1.4 The Engineering Design Process; 2. PROJECT MANAGEMENT FUNDAMENTALS, by C.C. Tonies: 2.1 Introduction, 2.2 Overview of a Software Project, 2.3 Project Planning and Management Philosophy, 2.4 Causes and Control of Entropy; 3. SOFTWARE DESIGN, by J.C. Enos, R.L. Van Tilburg: 3.1 The Human Problem-Solving Process, 3.2 Software Design to Produce a Product, 3.3 Software Design and System Engineering, 3.4 The Structuring Process, 3.5 Continuous Verification and Validation, 3.6 Summary; 4. STRUCTURED PROGRAMMING, by R.W. Jensen: 4.1 Introduction, 4.2 History and Background, 4.3 Theory and Techniques, 4.4 Implementations in High-Order Languages; 5. VERIFICATION AND VALIDATION, by M.S. Deutsch: 5.1 Definition of Terms, 5.2 Software Testing Methodologies, 5.3 Automated Testing, 5.4 Testing Real-Time Systems, 5.5 V and V Over the Software Life Cycle, 5.6 Future Trends; 6. SECURITY AND PRIVACY, by P.A. Hascall: 6.1 Level of Protection, 6.2 Chapter Overview, 6.3 Operating Systems, 6.4 Authentication, 6.5 Cryptography, 6.6 Confinement, 6.7 User Programmable Security Techniques, 6.8 Privacy, 6.9 Security Measurement, 6.10 Summary; 7. LEGAL ASPECTS OF SOFTWARE DEVELOPMENT, by C.H. Reddien: 7.1 Introduction, 7.2 Considerations of Selecting a Business Form, 7.3 Legal Considerations of Setting Up a Software Business, 7.4 Liabilities for Software Performance, 7.5 Other Legal Considerations; APPENDIX: Software Engineering Education - A Constructive Criticism.

Project management

• Donaldson, H., A GUIDE TO THE SUCCESSFUL MANAGEMENT OF COMPUTER PROJECTS, Associated Business Press, London, 1978, pp.xi + 266

"[...] This book is the result of the author's many years experience with computer projects and its simple purpose is to equip the data processing manager or the line manager for the task of implementing from scratch a computer project systematically and successfully. It provides a

complete and authoritative text which covers all aspects of managing the computer project - both mainframe and minicomputer, both batch and interactive computing. The approach is practical and direct and is based on the belief that the problems of computer project development can be solved through a style of management that allows for the delegation of tasks. At the same time the author recognises that the project team works best within an organised and stable atmosphere where ground rules are clearly established in advance -there being not too many ways of delivering a high quality product on time and within budget. From the universe of possible management styles and project methods the author describes the (usually simple) ones that work in practice. Examples and case-studies from real life situations are used extensively in the text to illustrate the author's points and themes. But the book is also sufficiently comprehensive and universal in its applications to be adopted as a 'standards manual' within a computer installation.[...] " (copertina).

Indice: 1. A Framework for Project Management, 2. The Project Management Cycle, 3. Project Planning Case Study, 4. Project Documentation and Reporting, 5. Survey Stage of Project Development, 6. The Evaluation Stage of Project Development, 7. Systems Design Strategy, 8. Specification Stage of Project Development, 9. Control, Security and Privacy, 10. Programming Stage of Project Development, 11. Program and System Testing, 12. Implementation Stage of Project Development, 13. Writing Procedure Manuals, 14. Operating Procedures, 15. System Maintenance and Review, 16. Postscript.

Walkthroughs

● Yourdon, E., STRUCTURED WALKTHROUGHS, Prentice-Hall Software Series, Prentice-Hall Inc., 1979, pp.xi + 137

"[...] This book has three objectives: The first is to acquaint you with the concept of walkthroughs, and the related concept of teams. In Chapter 2, we will discuss a number of different *types* of walkthroughs, and see where each type fits into a typical development life cycle for data processing projects. In Chapter 3, we will discuss the concept of programming teams - sometimes known as "egoless teams" - and their relationship with structured walkthroughs. A second, and more important objective, is to provide you with guidelines and procedures for successful implementation of walkthroughs in your organization. Chapter 4 through 7 discuss roles of different people in a walkthrough, activities that should be carried out before a walkthrough takes place, the conduct of the walkthrough itself and the activities after the walkthrough. A third objective of the book is to acquaint you with the psychological problems you are likely to encounter with other technicians, and with data processing management. Peer group reviews are not always as simple, friendly and objective as one would hope; and *programmers* are not always as rational, even-tempered and good-natured as one would hope. The situation is further complicated by EDP management, who are concerned about the impact that walkthroughs will have on their organization. Chapter 8 and 9 are about the *psychology* of walkthroughs. These two chapters were written primarily for programmers, analysts and other technicians - but it certainly would not hurt managers to read these chapters to gain an appreciation of the problems that technicians face. Conversely, Chapter 10 through 13 were written primarily for project managers - but it wouldn't hurt technicians to read the chapters. The more both groups understand about their respective roles, responsibilities and problems, the more likely that walkthroughs can be successfully implemented.[...]"

Indice: Part 1: INTRODUCTION: 1 Introduction, 2 Types of Walkthroughs, 3 Programming Teams; Part 2: THE MECHANICS OF WALKTHROUGHS: 4 Roles in a Walkthrough, 5 Activities Before the Walkthrough, 6 Activities During the Walkthrough, 7 Activities After the Walkthrough; Part 3: THE PSYCHOLOGY OF WALKTHROUGHS: 8 Observations About Programmers and Analysts, 9 Games Programmers Play; Part 4: MANAGEMENT'S ROLE IN WALKTHROUGHS: 10 Convincing Managements Walkthroughs Pay Off, 11 Avoiding the Urge to Attend Walkthroughs, 12 Performance Evaluation in Walkthroughs, 13 Building Teams.

Software standards

● Tausworthe, R.C., STANDARDIZED DEVELOPMENT OF COMPUTER SOFTWARE, PART II - STANDARDS, Prentice-Hall Inc., 1979, pp.x + 548

"[...] At the time Part I of this work [vedi IL SEGNALIBRO, NOTE DI SOFTWARE 3] was being published, the Jet Propulsion Laboratory's Deep Space Network (DSN) was in the process of developing and writing a set of Software Standard Practices, for which Part I was cited as the

'methodology textbook'. The standards were developed over several years by a group we simply called the 'Software Seminar'.
[...] Part II of this monograph, then, exposes this detailed set of rules for software implementation. I have broadened some of the DSN practices in some instances, in an attempt to make them more readily adaptable to organizational structures different than that of the DSN. Additional consonant practices from other sources have also been incorporated to broaden the scope of applicability to projects of types other then the high-technology, high-efficiency, single-purpose, custom-built variety demanded by the deep-space-station environment.[...]''

Indice di questo volume (Part II): XI. SOFTWARE REQUIREMENTS AND DEFINITION STANDARDS: 11.1 Generating Software Requirements, 11.2 Generation of the Software Architectural Design, 11.3 Generating the Software Functional Specification, 11.4 Documenting Technical Requirements and Functional Specifications, 11.5 Rules for the Software Development Library; XII. PROGRAM DESIGN AND SPECIFICATION STANDARDS: 12.1 Rules for Structural Design, 12.2 Rules for Data Structuring and Resource Access Design, 12.3 Rules for Developing Structured Programs, 12.4 Rules for Applying Structured Programming Theory, 12.5 Rules for Real-Time Structured Programs, 12.6 Standard Design Practices, 12.7 Rules for Documenting Structured Specifications, 12.8 Rules for the Software Development Library; XII. PROGRAM CODING STANDARDS: 13.1 Rules for Coding Structured Programs, 13.2 Rules for Coding Structured Real-Time Programs, 13.3 Rules for Documenting Structured Code, 13.4 Standard Production Procedures; XIV. DEVELOPMENT TESTING STANDARDS: 14.1 Rules for Specifying Development Tests, 14.2 Rules for Developing Tests for Real-Time Programs, 14.3 Rules for Assembling and Performing Tests, 14.4 Rules for Coding Test Elements, 14.5 Rules for Documenting Development-Test Specifications, 14.6 Rules for Documenting Test Results, 14.7 Rules for the Software Development Library, 14.8 Diagnostic Procedures; XV. QUALITY ASSURANCE STANDARDS: 15.1 Standard QA Activities, 15.2 QA Measures During Program Development, 15.3 Software Testing Characteristics, 15.4 Rules for Acceptance Testing and Certification, 15.5 Software Audits, 15.6 Documentation of QA Activities, 15.7 Rules for Security, Integrity, and Configuration Control; XVI. LEVELS OF DOCUMENTATION: 16.1 Human Factors, 16.2 Documentation Standards, 16.3 Preparation of Documentation; XVII. A STANDARD SOFTWARE PRODUCTION SYSTEM: 17.1 An Integrated Software Production System, 17.2 The Standard Production System Support Library, 17.3 Standard Programming Languages and Language Standards, 17.4 CRISP-PDL Processing, 17.5 Flowcharting from CRISP-PDL, 17.6 Text and Program File Editing, 17.7 Management Data and Status Reporting, 17.8 Conclusion; APPENDICES: A. Glossary of Terms and Abbreviations, B. Standard Flowchart Symbols, C. Software Requirements Document Topics, D. Software Definition Document Outline, E. Software Specification Document Outline, F. User Instruction Manual Topics, G. CRISP Syntax and Structures, H. Development Project Notebook Contents, I. Operations Manual Contents, J. Software Test Report Contents, K. Software Maintenance Manual Contents, L. Sample Programs for Project Management, M. Useful Standard Forms.

Si noti che le appendici coprono, in dimensione, più della metà del volume.

Software Requirements

● Heninger, K.L., SPECIFYING SOFTWARE REQUIREMENTS FOR COMPLEX SYSTEMS: NEW TECHNIQUES AND THEIR APPLICATION, IEEE Transactions on Software Engineering, vol. SE-6, n. 1, gennaio 1980, 2-13

''This paper concerns new techniques for making requirements specifications precise, concise, unambiguous, and easy to check for completeness and consistency. The techniques are well-suited for complex real-time software systems; they were developed to document the requirements of existing flight software for the Navy's A-7 aircraft. The paper outlines the information that belongs in a requirements document and discusses the objectives behind the techniques. Each technique is described and illustrated with examples from the A-7 document. The purpose of the paper is to introduce the A-7 document as a model of a disciplined approach to requirements specification; the document is available to anyone who wishes to see a fully worked-out example of the approach.''

Tecniche di disegno

● Parnas, D.L., DESIGNING SOFTWARE FOR EASE OF EXTENSION AND CONTRACTION, IEEE Transactions on Software Engineering, vol. SE-5, n. 2, marzo 1979, 128-138

"Designing software to be extensible and easily contracted is discussed as a special case of design for change. A number of ways that extension and contraction problems manifest themselves in current software are explained. Four steps in the design of software that is more flexible are then discussed. The most critical step is the design of a software structure called the "uses" relation. Some criteria for design decisions are given and illustrated using a small example. It is shown that the identification of *minimal* subsets and *minimal* extensions can lead to software that can be tailored to the needs of a broad variety of users."

Un nuovo libro sulla programmazione strutturata

● Linger, R.C., Mills, H.D., Witt, B.I., STRUCTURED PROGRAMMING: THEORY AND PRACTICE, Addison-Wesley Publishing Company, 1979, pp. xiv + 402

"The objective of this book is to show practicing programming professionals how to be more powerful, how to design more reliable and efficient software by the use of systematic methods of program analysis and synthesis. The central theme of these methods is the mathematical correctness of programs. There are two important by-products of this theme; namely, 1) the discipline of mathematical correctness provides a check and balance for the free and creative inventions that are so necessary in software design, and 2) the ability to create logically correct designs can be parlayed into actual programs that require little or no debugging. [...] This book is dedicated first to the new programmer beginning a professional career in software engineering, who already knows how to program computers and presumably has been well taught in structured programming from the beginning. But a beginning programmer needs a deeper foundation to cope with pressures that lie ahead - pressures from the complexities of ill-defined problems and poorly conceived software development tools, and most of all, pressures from large-scale, difficult deadline projects. All of these pressures will cry out for shortcuts and compromises. But many of these siren calls are pitfalls that lead to more difficulties and frustrations that can be imagined. Without strong inner discipline, based on a deep understanding of how to deal with massive logical designs and their complexities, the best of intentions and techniques are soon swamped. [...] This book is also dedicated to the professional development of those more-experienced programmers who have already made or are about to make the transition to structured programming. [...] Finally, this book is dedicated to programming managers. The management of programming projects is a difficult and rewarding job. But the lack of sound technical direction is as disastrous as the lack of good organization or personnel motivation in programming management. This book can help with the technical part of the management problem."

Indice: 1 Precision Programming, 2 Elements of Logical Expression, 3 Elements of Program Expression, 4 Structured Programs, 5 Reading Structured Programs, 6 The Correctness of Structured Programs, 7 Writing Structured Programs.

Winograd: un punto di vista

● Winograd, T., BEYOND PROGRAMMING LANGUAGES, Communications of the ACM, vol. 22, n. 7 (luglio 1979), 391-401

"As computer technology matures, our growing ability to create large systems is leading to basic changes in the nature of programming. Current programming language concepts will not be adequate for building and maintaining systems of the complexity called for by the tasks we attempt. Just as high level languages enabled the programmer to escape from the intricacies of a machine's order code, higher level programming systems can provide the means to understand and manipulate complex systems and components. In order to develop such systems, we need to shift our attention away from the detailed specification of algorithms, towards the description of the properties of the packages and objects with which we build. This paper analyzes some of the shortcomings of programming languages as they now exist, and lays out some possible directions for future research."

Sistemi informativi

● Langefors, B., INFOLOGICAL MODELS AND INFORMATION USER VIEWS, Information Systems, vol. 5, n. 1, 1980, 17-32

"The study of data systems as information systems put into focus the role of data as representation of information in the sense of knowledge about some slice of the world. This information system-view-

or infological view-made it clear that the data alone cannot "carry" information. They can only, at best, give rise to information in the minds of people and only in those people who hold a suitable frame-of-reference or world view, or "receiving structure", in their mind. Thus the infological perspective had to be widened successively from a concern merely with information representation, structuring, and exploitation to the study of social, socio-psychological and socio-linguistical aspects and of "object system", job design and other socio-technical issues.
The term "user view" is employed widely in recent data base work. The use of the term "user view" suggests such an "infological" perspective. However, a closer look at how the term is actually used indicates a much more delimited interpretation which focuses on representational aspects and processing. This is also made explicit, to some degree, by the usual formulation "user view of the data" rather than, for instance, "user view of the world".
In this paper a brief study is made of two aspects of user views,
(i) the *infological/conceptual aspect*, which is concerned with how conception relates to data and information, and to reality, and
(ii) the *"datalogical" aspect*, which is concerned with the selection of data from a data base and the rearrangement of them to suit a "user view" of the data (as seen from the application programmer).
The infological aspect is illustrated through a discussion of some of my own earlier results which are here brought together. The datalogical aspect is exemplified by some quotations from the most recent data base literature, as well as by some earlier results of my own.
The term "user view" if frequently used in the datalogical sense whereas the infological or information-system-theoretical studies often have addressed questions that have to do with the infological/conceptual aspects of "user views", without employing that term. In this paper some aspects of the problem of infological/conceptual user views are treated with a view to gain understanding of how both aspects of user views affect the design and use of information systems and data bases. Illustrations are taken from the author's own earlier work, for three reasons: (i) they were most easily available, (ii) they are directly associated with the problem at hand, and (iii) they have earlier been scattered over several works and it was useful, for the purpose of the present discussion, to bring them together."

Un libro sull'automazione d'ufficio

● Uhlig, R.P., Farber, D.J., Bair, J.H., THE OFFICE OF THE FUTURE, North-Holland Publishing Co., Monograph Series of the International Council for Computer Communications, vol. 1, 1979, pp. xiv + 379

Indice: PART I USES OF COMPUTERS IN THE OFFICE OF THE FUTURE: I.1 Introduction, I.2 The Automated Office of the Future, I.3 Tools to Support the Communication Activity, I.4 Text Editing Tools for Generating, Organizing, Analyzing and Transforming Information, I.5 The Integration of Computer Based Tools, I.6 Data Gathering and Information Retrieval Tools, I.7 Coordination Tools in the Office of the Future, I.8 Tools to Support Office Processes, I.9 Organization to Implement Interactive Computer Based Office Support Systems, I.10 Integration of the Spoken Word with Interactive Computer Based Office Support Systems; PART II TECHNOLOGICAL IMPERATIVES: II.1 The Underlying Technology, II.2 The Revolution - Communications, II.3 The Case for Distributed Processing, II.4 The Digital Channel - The Evolving Choice for Communications, II.5 Software Aspects of the Office Environment, Utilizing the Micro-processor, II.6 Distributed Processing, II.7 Distributed Data Bases, II.8 An Overview of Two Systems - One of the Recent Past, One of the Future, Conclusion: What is next; PART III THE IMPACT OF OFFICE AUTOMATION: III.1 Introduction, III.2 A Strategy for the Implementation of Office Automation Systems, III.3 Methods of Assessment and of Understanding Impact, III.4 Impacts on the Organization, III.5 Impacts on Groups, III.6 Impacts on the Individual, III.7 Economic Payoffs: the Productivity Question, III.8 Conclusion: A Recommendation.

Ada

● PRELIMINARY ADA REFERENCE MANUAL, SIGPLAN Notices, vol. 14, n. 6, june 1979, Part A

● Ichbiah, J.D., Barnes, J.G.P., Heliard, J.C., Krieg-Brueckner, B., Roubine, O., Wichmann, B.A., RATIONALE FOR THE DESIGN OF THE ADA PROGRAMMING LANGUAGE, SIGPLAN Notices, vol. 14, n. 6, june 1979, Part B

"This document, the Rationale for the design of the Green programming language, and the companion Reference Manual, are the two defining documents for the Green language. They serve different purposes. The Reference Manual contains a complete and concise definition of the language. Following Wirth we believe in the virtue of having a rather short reference manual. This has the advantage of providing the information in a form that can easily be consulted, read and reread several times, as the basis for developing a good familiarity with the language. Having a short reference manual however does not permit the infusion of much motivational information, and more generally, of information describing the reasons behind the major design decisions. Neither does it permit the insertion of the larger examples that are yet essential to help the reader get a better feeling for the language, and for the interaction among its features. The Rationale is meant to serve precisely that purpose. It is divided in chapters covering different aspects of the language. Most chapters of the Rationale correspond to chapters of the Reference Manual. Expressions and Statements are here regrouped in a single chapter, since the subject is fairly classical. Conversely, in view of the importance of the subjects, special chapters are devoted to numeric types and to access types, in addition to the chapter on types. All chapters of the Rationale are fairly independent (at the cost of some repetition) and can be read in any order after an initial pass over the Reference Manual. [...] A discussion of the technical issues follows the informal introduction. Such discussions cover the major design decisions, their justification, and the interactions with other aspects of the language. In particular these discussions consider implementability: designing a programming language without a deep concern for implementability would be little more than an exercise in style. [...]"

Costruzione di compilatori

● Brown, P.J., WRITING INTERACTIVE COMPILERS AND INTERPRETERS, John Wiley & Sons, 1979, pp. xvii + 265

"If you wish to implement an interactive language this book is aimed at you. It does not matter whether you are a hobbyist, a student, a professional systems programmer, or even a combination of all three of these. Nor does it matter if your motive is commercial gain, satisfying academic criteria or the sheer enjoyment of making something. The principles and techniques for doing a good job are the same for everybody. There are many existing books on implementing programming languages, and a lot of them are very good indeed. This book differs from the established literature in three ways. Firstly it deals with interactive languages, which demand different techniques and present different challenges from the traditional non-interactive languages. Secondly it is a practical book-it assumes you actually want to implement something, rather than study theoretical concepts; there is therefore as much material on planning and performing the task of implementing a language as there is on the underlying theoretical principles. Thirdly it aims to be a simple book, assuming no more from the reader than an ability to program and a familiarity with interactive working. If you are more knowledgeable, there are some sections you will be able to skip over. [...]"

Indice: Part 1: PLANNING: 1.1 Why Interactive?, 1.2 Planning Use of Resources, 1.3 Documentation, 1.4 Designing the Source Language and the User Interface, 1.5 Encoding the Compiler; Part 2: THE STRUCTURE OF A COMPILER: 2.1 Filling the Gaps, 2.2 Description of Terminology and Environment, 2.3 Source and Internal Languages, 2.4 Incremental Compiling, 2.5 Re-creating the Source Program, 2.6 Levels of Internal Language, 2.7 The Compilers, 2.8 Error Checking, 2.9 Error Messages, 2.10 Names, Scope and Data Types, 2.11 Dictionaries and Tables, 2.12 Storage Management, 2.13 The Editor, 2.14 Input and Output, 2.15 Break-ins, 2.16 Summary of Design; Part 3: THE DESIGN OF AN INTERNAL LANGUAGE: 3.1 Reverse Polish Notation, 3.2 Operators, 3.3 Encoding Reverse Polish, 3.4 A Brief Summary; Part 4: THE TRANSLATOR: 4.1 Overall Translator Organization, 4.2 Lexical Analysis, 4.3 Grammars, 4.4 Using Grammars for Parsing, 4.5 Checking and Resolving Data Types, 4.6 Semantic Actions; Part 5: THE RUN-TIME SYSTEM: 5.1 Error Detection and Diagnosis, 5.2 Executing Reverse Polish, 5.3 Allocating and Referencing User Variables, 5.4 Execution of Statements, 5.5 String Temporaries; Part 6: OTHER MODULES: 6.1 The Pre-Run Module, 6.2 The Re-creator Module, 6.3 The Command Module; Part 7: TESTING AND ISSUING: 7.1 Testing the Compiler, 7.2 Issuing; Part 8: SOME ADVANCED AND SPECIALIZED TOPICS: 8.1 Some Special Compilers, 8.2 Dynamic Compiling; Summary of the Deadly Sins.

Precompilatori Fortran

● Meeson, R., Pyster, A., OVERHEAD IN FORTRAN PREPROCESSORS, SOFTWARE -Practice and Experience, vol. 9, n. 12, dicembre 1979, 987-999

The quality of code generated by two commercially available FORTRAN preprocessors is examined. These preprocessors augment FORTRAN with 'structured' control structures such as the IF-THEN-ELSE and WHILE statements. Two versions of benchmark programs were written and tested, one version produced by the preprocessor, the other, performing the same computation, produced by careful hand-coding directly in FORTRAN. The code generated by the preprocessor and the code written directly in FORTRAN were then compiled. The resulting object modules were then compared for relative execution time and size. This experiment was repeated on three computers, and five compilers with various optimization levels. The results indicate that a substantial overhead in storage space may be paid by using a preprocessor rather than direct coding in FORTRAN, and that in some cases execution time may be increased somewhat by using a FORTRAN preprocessor.''

INDICE

Segue l'indice dei primi dodici numeri di NOTE DI SOFTWARE, realizzato con il sistema Kwoc descritto in altra parte di questo numero.

L'indice base è ordinato per anno, numero della rivista, e progressivo nell'ambito di tale numero; l'indice per autore è ordinato alfabeticamente.

CODICE INDICE BASE UNIVERSITA' DI MILANO

77 1 1 GARIBOLDI F. GIORDANI B. LANZARONE G.A. RENESTO P.V.
FLAGGER : UNO STRUMENTO PER LA VALUTAZIONE DEI RISULTATI DELLE PROVE

77 1 2 FARNEDI S. MAIOCCHI M. POGGIALI C. POLILLO R.
STNAL : UN INSIEME DI MACRO PER LA PROGRAMMAZIONE STRUTTURATA IN NAL

77 1 3 PARINI M.
IL PROBLEMA DELL' ACCESSO CONCORRENTE A UNA BASE DI DATI

77 1 4 NON FIRMATO
BIBLIOGRAFIA: LIBRI SULLA PROGRAMMAZIONE STRUTTURATA

77 1 5 NON FIRMATO
AVVENIMENTI: GIORNATA DI LAVORO SU "PROBLEMI DI UNIFORMIZZAZIONE DELLE STRUTTURE DI CONTROLLO PER LA PROGRAMMAZIONE STRUTTURATA IN FORTRAN E COBOL", MILANO, MARZO 1976

77 1 6 NON FIRMATO
BIBLIOTECA

77 2 1 CAJANI V. ROUSSEAU R.
MACRO PROCESSORS : GENERAL CONCEPTS

77 2 2 STOPPA C.
MACRO JCL, CONCETTI E APPLICAZIONI

77 2 3 BONESSO G. MAIOCCHI M. SPADONI D.
USO DI UN MACROPROCESSOR PER LINGUAGGI AD ALTO LIVELLO, COBRA (COBOL RAISING) UN'APPLICAZIONE IN COBOL

77 2 4 GOBBI G. TRABATTONI L.
MACRO E ASSERZIONI

77 2 5 GARIBOLDI F.
MACRO PER VALUTAZIONI DI COPERTURA DI TESTS

77 2 6 MONOPOLI M.L. PRIGIONE F.
USO DI UN MACROGENERATORE PER LA GENERAZIONE DI PROGRAMMI DIAGNOSTICI SOTTO SISTEMA OPERATIVO

77 2 7 TRABATTONI L.
L'USO DI MACRO JCL PER IL TESTING PERSONALIZZATO

77 2 8 BARBAGLIO C.
STRUTTURE DI DATI ASTRATTE IN ASSEMBLER MEDIANTE MACRO

77 2 9 FAIDUTTI F.
UN ESEMPIO DI MACRO PER GESTIRE STRUTTURE DI DATI IN ASSEMBLER : IL CASO DEGLI ALBERI

77 210 PRIGIONE F.
USO DI UN MACROGENERATORE NELLO SVILUPPO DI COMPILATORI FIRMWARE SPECIALIZZATI

77 211 ZAFFERRI G.
L' USO DI MACRO PER LA GESTIONE SEMIAUTOMATICA DI CONDIZIONI ECCEZIONALI : L' ESEMPIO DI DACBOL

77 212 MONDUCCI C. PARINI M.
STAND ALONE SYSTEM GENERATION LANGUAGE : LA GENERAZIONE DI UNA IMMAGINE DI MEMORIA ATTRAVERSO LA DEFINIZIONE DI MACRO

CODICE INDICE BASE

77 213 MAIOCCHI M. POLILLO R.
LA COSTRUZIONE DI MACROPROTOTIPI STRUTTURATI

77 214 BARBATO M.
AVVENIMENTI: HONEYWELL SOFTWARE PRODUCTIVITY SYMPOSIUM (APRILE 1977)

77 215 NON FIRMATO
IL SEGNALIBRO

77 3 1 DEGLI ANTONI G. TORRIANI G.
DOCUMENTAZIONE E PRODUTTIVITA'

77 3 2 ARMANINI G. CANDELA R. LEVA S. MARI R. MAZZOCCHI S.
LA REALIZZAZIONE DELLA DOCUMENTAZIONE PER UTENTI DEL LIVELLO 62

77 3 3 GOBBI G. MAIOCCHI M.
AUTODOCUMENTAZIONE DEI PROGRAMMI

77 3 4 MASERATI A. MAZZOCCHI S. POLILLO R. TORRIANI G.
DOCUMENTAZIONE PER L'UTENTE: ALCUNE INDICAZIONI E UN ESEMPIO

77 3 5 BECATTINI L. FAIDUTTI F.
VALUTAZIONE E MIGLIORAMENTO DELLE PRESTAZIONI DI STNAL

77 3 6 ANTONELLI L.
UN'OPPORTUNA STRUTTURA DI DATI PUO' SEMPLIFICARE LE STRUTTURE DI CONTROLLO?

77 3 7 CAZZIOL A. CUCCIATI C.
TMSS: UNO STRUMENTO PER LA GESTIONE AUTOMATICA DI CHECK-LIST DINAMICHE NELLA QUALIFICAZIONE INDUSTRIALE DEL SOFTWARE

77 3 8 ANSELMETTI W.
AVVENIMENTI: IL CONGRESSO ANNUALE DELL'AICA (1977)

77 3 9 NON FIRMATO
IL SEGNALIBRO

77 4 1 DEGLI ANTONI G. ZONTA B.
METAFORA E DOCUMENTAZIONE

77 4 2 CAPPELLI S. DE ANTONELLIS V. MAZZOCCHI S. POLILLO R.
SCHEMI DI DIALOGO: UNO STRUMENTO PER LA RAPPRESENTAZIONE DEI DIALOGHI IN UN SISTEMA INTERATTIVO

77 4 3 BETTINAZZI T. MAIOCCHI M. MASERATI A. MONZANI F. SPOLETINI E. TOSCANI E.
PHOS - UNA METODOLOGIA PER LA COSTRUZIONE VELOCE E AFFIDABILE DI PROGRAMMI

77 4 4 BETTINAZZI T. MASERATI A. MONZANI F. TOSCANI E.
PHOS - UNA APPLICAZIONE IN AMBIENTE EDP

77 4 5 CAVALLARI P. PEDROTTI T. TISATO F.
IL SIMULA 67 COME LINGUAGGIO DI SPECIFICA ESEGUIBILE

77 4 6 NON FIRMATO
IL SEGNALIBRO

78 5 1 BOYD D.L. PIZZARELLO A.
OVERVIEW OF WELLMADE DESIGN METHODOLOGY

78 5 2 ANTOCICCO S. BARASSI C. MAIOCCHI M. MARIGHELLA P.
SECOB : UN PRECOMPILATORE COBOL INTEGRATO PER LA PROGRAMMAZIONE STRUTTURATA

78 5 3 FISCHETTI E. PETRAGLIA G.
GENERAZIONE AUTOMATICA DI UNITA' DI PROGRAMMA: UN APPROCCIO

78 5 4 FERRARO G. MARINI D.
LA GENERAZIONE AUTOMATICA DEI DATI DI TEST

78 5 5 JEMOLI A.
UNO STRUMENTO PER LA GESTIONE DI ARCHIVI DI TEST

78 5 6 GALDI G. PETRAGLIA G. VILDACCI G.
SPECIALIZZAZIONE DI UN COMMAND LANGUAGE IN AMBIENTE BATCH REMOTO

78 5 7 CASTELLI G.
BIBLIOGRAFIA: IL LINGUAGGIO PASCAL NELLA LETTERATURA

78 5 8 CICU A. GHELFI P.
AVVENIMENTI: SUCCESSFUL SOFTWARE MANAGEMENT TECHNIQUES CONFERENCE (MARZO 1978)

78 5 9 NON FIRMATO
IL SEGNALIBRO

CODICE INDICE BASE

78 6/7 1 BRUNI F.
ASSEMBLATORI UNIVERSALI PER MICROPROCESSOR FUNZIONANTI SU MINI : STRUTTURA E PORTABILITA'

78 6/7 2 ARESI F. BIGAZZI A.
UN GENERATORE DI ASSEMBLATORI PORTABILI (RIASSUNTO)

78 6/7 3 CASTELLI G.
IMPLEMENTAZIONE DI LINGUAGGI IN FORTRAN : IL CASO DI MODULA

78 6/7 4 MASERATI A. MONZANI F.
PHOS : UNA METODOLOGIA PER LA COSTRUZIONE VELOCE ED AFFIDABILE DI PROGRAMMI (RIASSUNTO)

78 6/7 5 MANFREDI P. MARINI D.
VERSO LA PREVISIONE DELLO SFORZO DEL TESTING

78 6/7 6 DEGLI ANTONI G. MARINI D. PARINI M. ZONTA B.
LE RETI DI PETRI PER MODELLARE L'ATTIVITA' DI PRODUZIONE DEL SOFTWARE (RIASSUNTO)

78 6/7 7 DEGLI ANTONI G. MAIOCCHI M. POLILLO R.
PSPN : UN APPROCCIO AI PROBLEMI DI COMUNICAZIONE DELL' INFORMAZIONE TECNICA

78 6/7 8 CAVALLARI P. PEDROTTI T. TISATO F.
A PROGRAMMING METHODOLOGY USING SIMULA 67 LANGUAGE (RIASSUNTO ESTESO)

78 6/7 9 CERVETTO G.C.
IL PERFORMA COME PROPOSTA DI UN METODO ORIENTATO AI PROBLEMI APERTI DALLA PERFORMANCE EVALUATION : ANALISI E SINTESI (RIASSUNTO)

78 6/710 MARTUCCI R.
PROBLEMATICHE PER LA PROGRAMMAZIONE DEI SISTEMI DI TELECOMUNICAZIONE : CHILL COME PROPOSTA DI STANDARD INTERNAZIONALE E PLEO , L' ATTUALE SOLUZIONE SITS (RIASSUNTO)

78 6/711 ALBANI A.
DALLA S. MONDUCCI C.
LA PRODUZIONE DI PROGRAMMI DIAGNOSTICI
STRUMENTI DI MICROPROGRAMMAZIONE NELLO SVILUPPO DI GRANDI SISTEMI

78 6/712 SAVORETTI F.
UN GENERATORE AUTOMATICO DI TESTS PER COMPILATORI (RIASSUNTO ESTESO)

78 6/714 DE CINDIO F.
AVVENIMENTI: ECOLE D'ETE D'INFORMATIQUE "METHODOLOGIE DE LA PROGRAMMATION THEORIE ET PRATIQUE"

78 6/715 NON FIRMATO
IL SEGNALIBRO

78 8 1 CAJANI V.
INTRODUZIONE AL TPS

78 8 2 PARODI F. ZANONE G. FIRPO C. CALVINI M.
UNA ESPERIENZA DI PERFORMANCE EVALUATIOM IN ANSALDO

78 8 3 SARTORI L.
ARCHITETTURA TAGGED : PUO' L' HARDWARE CONTRIBUIRE ALLA AFFIDABILITA' DEL SOFTWARE ?

78 8 4 GUALZETTI M. MOTTA M. SQUIZZATO L.
STRUMENTI PER LA MICROPROGRAMMAZIONE STRUTTURATA

78 8 5 NON FIRMATO
IL SEGNALIBRO

79 9 1 SCURATTI C.
TESTGE: UNA TECNICA PER LA GENERAZIONE DI DATI DI PROVA AFFIDABILI ED ESAURIENTI PER APPLICAZIONI EDP

79 9 2 DEGLI ANTONI G. MARINI D. PARINI M. ZONTA B.
LE RETI DI PETRI NELLA DESCRIZIONE DI PROCEDURE ORIENTATE ALLA PRODUZIONE DI SOFTWARE

79 9 3 CIUCCI G. GUARNIERI A. QUARTAPELLE L.
BOOLEAN CONTROL OF PROGRAMMING: COME AUMENTARE IL GRADO DI AFFIDABILITA' DI PROCEDURE A SCARSA CONTROLLABILITA' INTERMEDIA

79 9 4 PEREGO G.
APPROCCI ALLA COMUNICAZIONE DI ATTIVITA' CONCORRENTI IN SISTEMI DI ELABORAZIONE DISTRIBUITI

79 9 5 DE CINDIO F.
AVVENIMENTI: INFORMATION SYSTEMS, ORGANIZATIONAL CHOICE, SOCIAL VALUES, PISA, APRILE 1979

79 9 6 CICU A.
AVVENIMENTI: TERZO CONVEGNO SULLA PROGRAMMAZIONE STRUTTURATA, MILANO, MAGGIO 1979

79 9 7 CICU A.
AVVENIMENTI: TERZA CONFERENZA INTERNAZIONALE HONEYWELL SULL' INGEGNERIA DEL SOFTWARE, BLOOMINGTON, MARZO 1979

CODICE INDICE BASE

79 9 8 NON FIRMATO
IL SEGNALIBRO

79 10/11 1 CASTELLI G. HAUS G. MAIOCCHI M.
UNA METODOLOGIA DISTESURA E VERIFICA DI SPECIFICHE FUNZIONAALI DI PROCEDURE E PROGRAMMI SOFTWARE

79 10/11 2 ROSSI G.P. SIMONE C.
SCHEMI DI INTERAZIONE FRA PROCESSI CONCORRENTI DESCRITTI MEDIANTE LE RETI DI PETRI

79 10/11 3 ROSSI G.P. SIMONE C.
PROCESSI CONCORRENTI IN UN SISTEMA MULTICOMPUTERS: DESCRIZIONE CON RETI DI PETRI

79 10/11 4 BELTRAMINI A.
UNA METODOLOGIA PER IL PROGETTO DI SISTEMI DI CONTROLLO DI PROCESSO BASATA SULLE RETI DI PETRI

79 10/11 5 ZONTA B.
AVVENIMENTI: ADVANCED COURSE ON GENERAL NET THEORY OF PROCESSES AND SYSTEMS, AMBURGO 8-19 OTTOBRE 1979

79 10/11 6 NON FIRMATO
IL SEGNALIBRO

79 12 1 CASTELLI G.
ALCUNI LUNGUAGGI A CONFRONTO

79 12 2 ALBANO A. MARIETTI A. OCCHIUTO M.E. ORSINI R.
LA GENERAZIONE DI RAPPORTI DA BASI DI DATI RELAZIONALI

79 12 3 LE VAN HUU
UN SISTEMA AD INDICE A DUE LIVELLI

79 12 4 BOSSAVIT A. MEYER B.
SUR LA PROGRAMMATION RATIONNELLE DES ALGORITHMES NUMERIQUES

79 12 5 DE MICHELIS G.
AVVENIMENTI: NOTE A MARGINE DEL WORKSHOP "STANDARDS FOR SYSTEM ANALYSIS", SVOLTOSI A ROMA NEL MARZO 1980

79 12 6 NON FIRMATO
IL SEGNALIBRO

INDICE PER AUTORE

	CODICE
ALBANI A.	78 6/711
ALBANO A.	79 12 2
ANSELMETTI W.	77 3 8
ANTOCICCO S.	78 5 2
ANTONELLI L.	77 3 6
ARESI F.	78 6/7 2
ARMANINI G.	77 3 2
BARASSI C.	78 5 2
BARBAGLIO C.	77 2 8
BARBATO M.	77 214
BECATTINI L.	77 3 5
BELTRAMINI A.	79 10/11 4
BETTINAZZI T.	77 4 3
	77 4 4
BIGAZZI A.	78 6/7 2
BONESSO G.	77 2 3
BOSSAVIT A.	79 12 4
BOYD D.L.	78 5 1
BRUNI F.	78 6/7 1
CAJANI V.	77 2 1
	78 8 1
CALVINI M.	78 8 2
CANDELA R.	77 3 2
CAPPELLI S.	77 4 2
CASTELLI G.	78 5 7
	78 6/7 3
	79 12 1
	79 10/11 1
CAVALLARI P.	77 4 5
	78 6/7 8
CAZZIOL A.	77 3 7
CERVETTO G.C.	78 6/7 9
CICU A.	78 5 8
	79 9 6
	79 9 7
CIUCCI G.	79 9 3
CUCCIATI C.	77 3 7
DALLA S.	78 6/711
DE ANTONELLIS V.	77 4 2
DE CINDIO F.	78 6/714
	79 9 5
DE MICHELIS G.	79 12 5
DEGLI ANTONI G.	77 3 1
	77 4 1
	78 6/7 6
	78 6/7 7
	79 9 2
FAIDUTTI F.	77 2 9
	77 3 5
FARNEDI S.	77 1 2
FERRARO G.	78 5 4
FIRPO C.	78 8 2
FISCHETTI E.	78 5 3
GALDI G.	78 5 6
GARIBOLDI F.	77 1 1
	77 2 5
GHELFI P.	78 5 8
GIORDANI B.	77 1 1
GOBBI G.	77 2 4
	77 3 3
GUALZETTI M.	78 8 4
GUARNIERI A.	79 9 3
HAUS G.	79 10/11 1
JEMOLI A.	78 5 5
LANZARONE G.A.	77 1 1
LE VAN HUU	79 12 3
LEVA S.	77 3 2
MAIOCCHI M.	77 1 2
	77 2 3
	77 213
	77 3 3
	77 4 3
	78 5 2
	78 6/7 7
	79 10/11 1
MANFREDI P.	78 6/7 5
MARI R.	77 3 2
MARIETTI A.	79 12 2
MARIGHELLA P.	78 5 2
MARINI D.	78 5 4
	78 6/7 5
	78 6/7 6
	79 9 2
MARTUCCI R.	78 6/710
MASERATI A.	77 3 4
	77 4 3
	77 4 4
	78 6/7 4
MAZZOCCHI S.	77 3 2
	77 3 4
	77 4 2
MEYER B.	79 12 4
MONDUCCI C.	77 212
	78 6/711
MONOPOLI M.L.	77 2 6
MONZANI F.	77 4 3
	77 4 4
	78 6/7 4
MOTTA M.	78 8 4
NON FIRMATO	77 1 4
	77 1 5
	77 1 6
	77 215
	77 3 9
	77 4 6
	78 5 9
	78 8 5
	78 6/715
NON FIRMATO	79 9 8
	79 12 6
	79 10/11 6
OCCHIUTO M.E.	79 12 2
ORSINI R.	79 12 2
PARINI M.	77 1 3
	77 212
	78 6/7 6
	79 9 2
PARODI F.	78 8 2
PEDROTTI T.	77 4 5
	78 6/7 8
PEREGO G.	79 9 4
PETRAGLIA G.	78 5 3
	78 5 6
PIZZARELLO A.	78 5 1
POGGIALI C.	77 1 2
POLILLO R.	77 1 2
	77 213
	77 3 4
	77 4 2
	78 6/7 7
PRIGIONE F.	77 2 6
	77 210
QUARTAPELLE L.	79 9 3
RENESTO P.V.	77 1 1
ROSSI G.P.	79 10/11 2
	79 10/11 3
ROUSSEAU R.	77 2 1
SARTORI L.	78 8 3
SAVORETTI F.	78 6/712
SCURATTI C.	79 9 1
SIMONE C.	79 10/11 2
	79 10/11 3
SPADONI D.	77 2 3
SPOLETINI E.	77 4 3
SQUIZZATO L.	78 8 4
STOPPA C.	77 2 2
TISATO F.	77 4 5
	78 6/7 8
TORRIANI G.	77 3 1
	77 3 4
TOSCANI E.	77 4 3
	77 4 4
TRABATTONI L.	77 2 4
	77 2 7
VILDACCI G.	78 5 6
ZAFFERRI G.	77 211
ZANONE G.	78 8 2
ZONTA B.	77 4 1
	78 6/7 6
	79 9 2
	79 10/11 5

Honeywell

Honeywell Information Systems Italia

Via Laboratori Olivetti - Pregnana Milanese

UNIVERSITA'
DEGLI STUDI DI MILANO
ISTITUTO DI CIBERNETICA
Via Viotti, 5 - Milano